Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: **Розробка мобільного застосунку навчання гри на гітарі з використанням штучного інтелекту**

Виконав(ла): студент(ка) IV курсу, групи СП-42
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

		Чайківський С.І.
	(підпис)	(прізвище та ініціали)
Керівник		Цебрій О.Р.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Стоянов Ю.М.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М.Р.
	(підпис)	(прізвище та ініціали)
Рецензент		
	(підпис)	(прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Чайківському Святославу Івановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка мобільного застосунку навчання гри на гітарі з використанням штучного інтелекту

Керівник роботи Цебрій Олексій Романович к. ф.-м. н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Предметна область, завдання, вимоги та специфікація, програмне рішення, методичні вказівки

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз вимог до програмної системи.
2. Проєктування та розробка програмної системи
3. Тестування, впровадження та підтримка системи.
4. Безпека життєдіяльності, основи охорони праці.
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Слайди презентації та діаграми процесів
1. Тема роботи. 2. Актуальність, мета, задачі дослідження. 3. Огляд існуючих застосунків для навчання гри на гітарі.
4. Функціональні та нефункціональні вимоги до системи. 5. Загальна архітектура застосунку (мультимодульна Clean Architecture).
6. Діаграма варіантів використання. 7. UML-діаграми (компонентів, класів, бази даних).
8. Архітектура AI-тьютора: RAG-пайплайн, Qdrant, LangGraph. 9. Модуль розпізнавання акордів (ChordMini).
10. Інтерфейси застосунку Guitarly. 11. Результати тестування. 12. Висновки по роботі.
13. Слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			
Нормоконтроль			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*6.04 – 12.04*	
2.	*Робота над першим розділом «Аналіз вимог до програмної системи»*	*13.04 – 26.04*	
3.	*Робота над другим розділом «Проєктування та розробка програмної системи»*	*27.04 – 03.05*	
4.	*Робота над третім розділом «Тестування, впровадження та підтримка системи»*	*04.05 – 17.05*	
5.	*Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»*	*18.05 – 24.05*	
6.	*Оформлення пояснювальної записки і графічного матеріалу*	*25.05 – 7.06*	
7.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*	
8.	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*	
9.	*Захист кваліфікаційної роботи бакалавра*		

Студент _____ Чайківський С.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Цебрій О.Р.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота бакалавра, виконав Чайківський Святослав Іванович, студент групи СП-42 Тернопільського національного технічного університету імені Івана Пулюя, на тему «Розробка мобільного застосунку для навчання гри на гітарі з використанням штучного інтелекту». Робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків і має загальний обсяг 107 сторінок, включає 42 рисунки, 13 таблиць, 4 додатки та бібліографію з 37 джерел.

Метою роботи є проектування та розробка мобільного застосунку "Guitarly" для платформи Android, що забезпечує інтерактивне навчання гри на гітарі з використанням технологій штучного інтелекту, зокрема методу генерації з доповненням пошуком (RAG), оркестрації агентів за допомогою LangGraph та автоматичного розпізнавання акордів у реальному часі.

У роботі реалізовано мультимодульний Android-застосунок на основі принципів Clean Architecture з використанням Kotlin та Jetpack Compose. Розроблено AI-тьютор на основі RAG-пайплайну з векторною базою даних Qdrant та оркестрацією LangGraph, що забезпечує персоналізовані відповіді на запитання щодо теорії гітарної гри у режимі стрімінгу через Server-Sent Events (SSE). Реалізовано модуль автоматичного розпізнавання акордів у YouTube-відео за допомогою спеціалізованої моделі ChordMini. Розроблено бібліотеку акордів з графічними діаграмами аплікатур, цифровий метроном, хроматичний тюнер та систему челленджів. Проведено модульне та інтеграційне тестування ключових компонентів системи.

Ключові слова: мобільний застосунок, Android, штучний інтелект, Kotlin, Jetpack Compose, RAG, LangGraph, Qdrant, ChordMini, навчання гри на гітарі, акорди, SSE-стрімінг, Clean Architecture.

ABSTRACT

Bachelor's qualification thesis, completed by Chaikivskyi Sviatoslav Ivanovich, a student of group SP-42 at Ternopil Ivan Puluj National Technical University, is devoted to "Development of a Mobile Application for Learning to Play Guitar Using Artificial Intelligence". The thesis consists of an introduction, four chapters, conclusions, a list of references, and appendices, with a total volume of 107 pages, including 42 figures, 13 tables, 4 appendices, and a bibliography of 37 sources.

The aim of the thesis is to design and develop the "Guitarly" mobile application for the Android platform, providing interactive guitar learning using artificial intelligence technologies, particularly Retrieval-Augmented Generation (RAG), agent orchestration via LangGraph, and real-time automatic chord recognition.

The thesis presents a multi-module Android application built on Clean Architecture principles using Kotlin and Jetpack Compose. An AI tutor based on a RAG pipeline with the Qdrant vector database and LangGraph orchestration was developed, providing personalized responses to guitar theory questions via Server-Sent Events (SSE) streaming. A module for automatic chord recognition in YouTube videos using the specialized ChordMini model was implemented. A chord library with fingering diagrams, a digital metronome, a chromatic tuner, and a social challenges system for learner motivation were developed. Unit and integration testing of the key system components was conducted.

The developed system demonstrates the practical application of modern artificial intelligence methods in the field of mobile educational applications. The architectural solutions for integrating RAG and LangGraph with an Android client via SSE streaming can be adapted for other e-learning domains.

Keywords: mobile application, Android, artificial intelligence, Kotlin, Jetpack Compose, RAG, LangGraph, Qdrant, ChordMini, guitar learning, chords, SSE streaming, Clean Architecture.

ЗМІСТ

ВСТУП 10
1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ 13
1.1 Аналіз предметної області 13
1.1.1 Огляд існуючих рішень 14
1.1.2 Виявлені проблеми та обмеження 17
1.2 Постановка завдання та цілей 19
1.3 Пошук акторів та варіантів використання 20
1.4 Опис ключових варіантів використання 21
1.5 Функціональні вимоги 23
1.6 Нефункціональні вимоги 24
1.7 Вибір та обґрунтування технологічного стеку 26
1.7.1 Мобільний застосунок (Android) 26
1.7.2 Серверна частина 28
1.7.3 Компоненти штучного інтелекту 29
1.8 Висновки до розділу 1 31
2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ 33
2.1 Вибір процесу розробки 33
2.2 Проектування архітектури системи 34
2.2.1 Загальна архітектура системи 34
2.2.2 Мультимодульна структура проекту 35
2.2.3 Clean Architecture та шари застосунку 38
2.2.4 Серверна інфраструктура 39
2.3 Побудова схем бази даних 40

2.4 UML-моделювання системи ... 42
2.4.1 Діаграми класів ... 42
2.4.2 Діаграми послідовності ... 45
2.4.3 Діаграми активності ... 47
2.4.4 Діаграми станів ... 49
2.5 Реалізація основних класів та методів ... 51
2.5.1 Модуль AI-тьютора ... 51
2.5.2 RAG-конвеєр та векторна база знань ... 52
2.5.3 Граф LangGraph та Self-RAG оркестрація ... 55
2.5.4 SSE-стрімінг відповідей ... 58
2.5.5 Розпізнавання акордів YouTube-відео ... 59
2.5.6 Взаємодія з ChordMini-сервісом ... 60
2.5.7 Обробка результатів на стороні Android ... 63
2.5.8 Бібліотека акордів ... 65
2.5.9 Тюнер та метроном ... 66
2.5.10 Модуль челенджів ... 67
2.5.11 Допоміжні feature-модулі ... 68
2.6 Розробка інтерфейсу користувача ... 69
2.7 Висновки до розділу 2 ... 70
3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА СИСТЕМИ ... 72
3.1 Тестування програмної системи ... 72
3.1.1 Види та план тестування ... 72
3.1.2 Розробка тестових сценаріїв ... 73
3.1.3 Навантажувальне тестування ... 74
3.2 Розгортання програмної системи та системні вимоги ... 77

3.2.1 Системні вимоги 77
3.2.2 Розгортання серверної частини 78
3.2.3 Розгортання Android-застосунку 79
3.3 Верифікація програмної системи 79
3.4 Висновки до розділу 3 80
4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 82
4.1 Працездатність людини-оператора в процесі розробки програмного забезпечення 82
4.2 Вимоги охорони праці до організації робочого місця користувача персонального комп'ютера 85
4.3 Висновки до розділу 4 89
ВИСНОВКИ 91
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 93
ДОДАТКИ 97

ПЕРЕЛІК СКОРОЧЕНЬ

AI – Artificial Intelligence (штучний інтелект)

API – Application Programming Interface (інтерфейс прикладного програмування)

CI/CD – Continuous Integration / Continuous Deployment (безперервна інтеграція / безперервне розгортання)

CPU – Central Processing Unit (центральний процесор)

DI – Dependency Injection (ін'єкція залежностей)

DL – Deep Learning (глибинне навчання)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

IDE – Integrated Development Environment (інтегроване середовище розробки)

JSON – JavaScript Object Notation (формат обміну даними JSON)

LLM – Large Language Model (велика мовна модель)

ML – Machine Learning (машинне навчання)

MVVM – Model-View-ViewModel (архітектурний шаблон модель-вигляд-модель вигляду)

ORM – Object-Relational Mapping (об'єктно-реляційне відображення)

RAG – Retrieval-Augmented Generation (генерація з доповненням пошуком)

RAM – Random Access Memory (оперативна пам'ять)

REST – Representational State Transfer (передача репрезентативного стану)

SDK – Software Development Kit (набір засобів розробки програмного забезпечення)

SQL – Structured Query Language (мова структурованих запитів)

SSE – Server-Sent Events (події, надіслані сервером)

UI – User Interface (інтерфейс користувача)

UML – Unified Modeling Language (уніфікована мова моделювання)

URL – Uniform Resource Locator (уніфікований локатор ресурсів)

UX – User Experience (досвід користувача)

ВСТУП

У сучасній галузі мобільних освітніх технологій, де персоналізація та доступність визначають ефективність навчання, застосування методів штучного інтелекту стає невід'ємною складовою розробки якісних програмних рішень. Гітара є одним із найпопулярніших музичних інструментів у світі. Досі мільйони людей щороку прагнуть опанувати її самостійно. Проте традиційні підходи до навчання мають суттєві обмеження: відсутність персоналізованого зворотного зв'язку, неможливість миттєво отримати відповідь на теоретичне питання та брак інструментів для аналізу власної гри.

Поява великих мовних моделей та методу генерації з доповненням пошуком (RAG) відкриває нові можливості для створення інтелектуальних тьюторів, здатних надавати точні й контекстно-залежні відповіді на запитання учня. Використання RAG-пайплайну з векторною базою даних дозволяє суттєво підвищити якість відповідей порівняно зі звичайними чат-ботами, оскільки модель спирається не лише на загальні знання, а й на спеціалізовані навчальні матеріали з теорії гітарної гри. Водночас оркестрація агентів за допомогою LangGraph забезпечує гнучку логіку обробки запитів: від пошуку та оцінки релевантності документів до виклику спеціалізованих інструментів: транспонування акордів, пошуку виконань та налаштування метронома.

Окремим напрямом є автоматичне розпізнавання акордів у музичному аудіо. Завдяки спеціалізованій моделі глибинного навчання ChordMini стає можливим аналіз YouTube-відео з визначенням послідовності акордів у реальному часі. Це усуває необхідність вручну шукати або записувати акорди до улюблених пісень, застосунок виконує цю роботу автоматично, суттєво прискорюючи процес навчання.

Реалізація зазначених технологій у рамках єдиного мобільного застосунку для Android, побудованого на принципах Clean Architecture та Jetpack Compose, дозволяє отримати масштабований і підтримуваний продукт, що поєднує AI-тьютора, розпізнавання акордів, бібліотеку аплікатур, цифровий метроном,

хроматичний тюнер та систему соціальних челленджів. Такий підхід є актуальною науково-практичною задачею, що відповідає сучасним тенденціям розвитку мобільних освітніх технологій.

Метою роботи є проєктування та розробка мобільного застосунку "Guitarly" для платформи Android, що забезпечує інтерактивне навчання гри на гітарі з використанням технологій штучного інтелекту, зокрема RAG, оркестрації агентів за допомогою LangGraph та автоматичного розпізнавання акордів у реальному часі.

Для досягнення мети необхідно вирішити такі задачі: провести аналіз предметної області та огляд існуючих рішень, виявити їх переваги й недоліки; сформулювати функціональні та нефункціональні вимоги до системи й побудувати модель варіантів використання; спроєктувати мультимодульний Android-застосунок на основі Clean Architecture з використанням Kotlin та Jetpack Compose; розробити AI-тьютора на основі RAG-пайплайну з векторною базою даних Qdrant та оркестрацією LangGraph із підтримкою SSE-стрімінгу відповідей; реалізувати модуль автоматичного розпізнавання акордів у YouTube-відео з використанням моделі ChordMini; розробити допоміжні feature-модулі: бібліотеку акордів з графічними аплікатурами, цифровий метроном, хроматичний тюнер та систему соціальних челленджів; провести модульне та інтеграційне тестування ключових компонентів системи і виконати її розгортання.

Об'єктом дослідження є процес навчання гри на гітарі з використанням мобільних технологій та штучного інтелекту.

Предметом дослідження є методи та засоби проєктування і реалізації мобільного застосунку з інтегрованим AI-тьютором на основі RAG та модулем автоматичного розпізнавання акордів.

Методи дослідження. У роботі використано методи порівняльного аналізу для дослідження існуючих рішень, UML-моделювання для проєктування архітектури, принципи Clean Architecture та Domain-Driven Design для організації кодової бази, методи векторного пошуку та RAG для побудови AI-тьютора, методи глибинного навчання для розпізнавання акордів, а також методи модульного та інтеграційного тестування для верифікації системи.

Практичне значення одержаних результатів. Розроблений застосунок "Guitarly" може використовуватись як самостійний продукт для вивчення гри на гітарі. Архітектурні рішення щодо інтеграції RAG та LangGraph з Android-клієнтом через SSE-стрімінг можуть бути адаптовані для створення AI-тьюторів в інших предметних областях електронного навчанн

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Перший розділ присвячено аналізу предметної області навчання гри на гітарі за допомогою мобільних технологій та штучного інтелекту. Розглядаються існуючі програмні рішення, виявляються їх недоліки, формулюються вимоги до розроблюваної системи та обґрунтовується вибір технологічного стеку.

1.1 Аналіз предметної області

Навчання гри на музичних інструментах є традиційно трудомістким процесом, що потребує регулярних занять з педагогом та самостійної практики. Гітара є одним з найпоширеніших інструментів у світі, за різними оцінками, понад 700 мільйонів людей грають на ній або мають бажання навчитись [3]. Популярність гітари зумовлена її доступністю, широким музичним діапазоном та застосуванням у різних жанрах від класики до рок-музики та джазу.

Розвиток мобільних технологій суттєво змінив підходи до самостійного навчання. Освітні застосунки дозволяють учням займатись у зручний час без прив'язки до розкладу викладача. Глобальний ринок мобільних освітніх застосунків у 2024 році оцінювався близько 70 млрд доларів США та продовжує зростати із середньорічним темпом (CAGR) близько 24% [2]. У сегменті навчання музиці з'явилась значна кількість застосунків, орієнтованих на різні рівні підготовки та стилі навчання.

Водночас більшість наявних рішень базуються на статичному контенті: відеоуроках, табулатурах та бібліотеках акордів, і не забезпечують інтерактивного зворотного зв'язку. Поява великих мовних моделей та методів RAG [4, 5] створює нові можливості для побудови персоналізованих AI-тьюторів, здатних відповідати на конкретні запитання учня у реальному часі. Паралельно розвиток моделей глибинного навчання для аналізу аудіо дозволяє автоматично розпізнавати акорди безпосередньо з відеозаписів [6], що усуває необхідність вручну шукати табулатури.

1.1.1 Огляд існуючих рішень

Для визначення місця розроблюваного застосунку на ринку проведено аналіз п'яти найбільш поширених рішень для навчання гри на гітарі. Критеріями порівняння обрано: наявність AI-тьютора, можливість автоматичного розпізнавання акордів, рівень персоналізації навчання, підтримувані платформи та модель монетизації. Результати порівняння наведено в таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих рішень для навчання гри на гітарі

Застосунок	**Платформа**	**AI-тьютор**	**Розпізнавання акордів**	**Персона-лізація**	**Монети-зація**
Yousician	iOS, Android, Desktop	Відсутній	Відсутнє	Часткова	Freemium
Ultimate Guitar	iOS, Android, Web	Відсутній	Відсутнє	Відсутня	Freemium
Fender Play	iOS, Android	Відсутній	Відсутнє	Часткова	Платний
Chordify	Web, iOS, Android	Відсутній	Так (аудіо)	Відсутня	Freemium
GuitarTuna	iOS, Android	Відсутній	Відсутнє	Відсутня	Freemium
Guitarly	Android	RAG+LangGraph	Так (YouTube/ChordMini)	Повна	Відкритий

Yousician (рис. 1.1) є одним з найпопулярніших застосунків для навчання гри на гітарі, що пропонує структуровані уроки з відеопоясненнями та ігрову механіку нарахування балів. Застосунок підтримує розпізнавання нот через мікрофон для оцінки точності гри, однак не має AI-тьютора та не дозволяє розпізнавати акорди з довільних відеозаписів. Доступ до повного функціоналу потребує платної підписки [8].

Рисунок 1.1 – Логотип Yousician

Ultimate Guitar (рис. 1.2) – найбільша бібліотека акордів та табулатур у світі, що налічує понад 1,1 мільйона пісень. Застосунок дозволяє переглядати акорди в режимі авто-прокрутки та транспонувати їх, проте весь контент є статичним: відсутній AI-тьютор, персоналізація та розпізнавання акордів з відео [9].

Рисунок 1.2 – Логотип Ultimate Guitar

Fender Play (рис. 1.3) пропонує структурований відеокурс навчання від компанії Fender, орієнтований на початківців. Уроки організовані за стилями музики та рівнями складності, проте застосунок повністю платний, не має AI-компонентів і не підтримує інтерактивний зворотний зв'язок [10].

Рисунок 1.3 – Логотип Fender Play

Chordify (рис. 1.4) спеціалізується на автоматичному розпізнаванні акордів з аудіо та відео. Система аналізує завантажений або вказаний через URL файл і відображає акорди, синхронізовані з відтворенням. Незважаючи на корисну функцію розпізнавання, застосунок не має освітньої складової, AI-тьютора та соціальних функцій [11].

Рисунок 1.4 – Логотип Chordify

GuitarTuna (рис. 1.5) є найбільш завантажуваним тюнером для гітари, який підтримує хроматичне та хордове налаштування. Застосунок вузькоспеціалізований і не претендує на роль повноцінного навчального середовища [12].

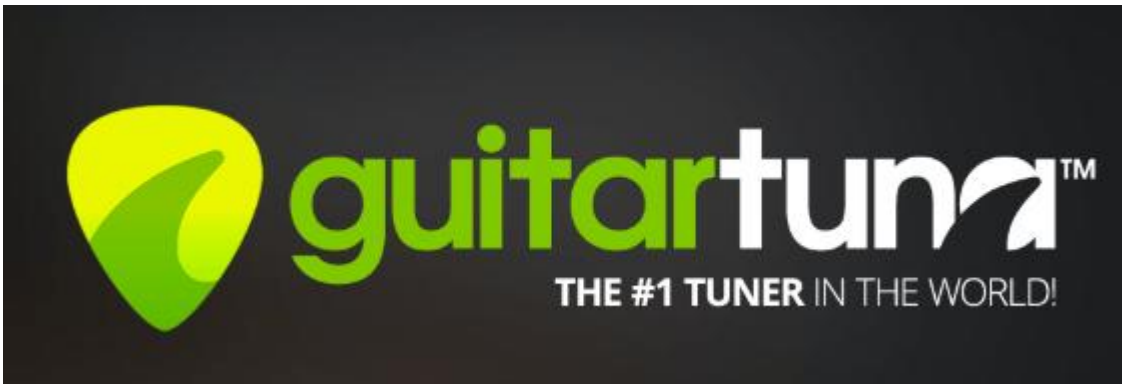


Рисунок 1.5 – Логотип GuitarTuna

Таким чином, жоден з розглянутих застосунків не поєднує персоналізованого AI-тьютора з функцією автоматичного розпізнавання акордів у YouTube-відео. Застосунок "Guitarly" заповнює цю нішу, інтегруючи RAG-тьютора на основі LangGraph, модуль ChordMini для розпізнавання акордів, бібліотеку аплікатур, метроном, тюнер та систему соціальних челленджів у єдиному Android-застосунку.

1.1.2 Виявлені проблеми та обмеження

Аналіз існуючих рішень дозволив виявити низку системних проблем, що обмежують ефективність самостійного навчання гри на гітарі за допомогою мобільних застосунків. Ці проблеми можна згрупувати за чотирма напрямами: відсутність інтелектуальної взаємодії, обмеження у роботі з відеоконтентом, слабка персоналізація та брак соціальної мотивації.

Відсутність персоналізованого AI-тьютора. Жоден з розглянутих застосунків не надає можливості поставити конкретне теоретичне запитання та отримати відповідь, що враховує контекст навчання. Учень, який стикається з незрозумілою технікою або теоретичною концепцією, змушений шукати відповідь у зовнішніх джерелах, форумах, відео на YouTube або у живого викладача. Це розриває процес

навчання та знижує ефективність самостійної практики. Наявні чат-боти загального призначення (ChatGPT, Gemini) не адаптовані до специфіки гітарної теорії та не мають доступу до спеціалізованих матеріалів [13].

Відсутність автоматичного розпізнавання акордів з відеозаписів. Одним із найпоширеніших сценаріїв самостійного навчання є вивчення улюбленої пісні за YouTube-відео. Проте жоден з розглянутих застосунків не дозволяє завантажити YouTube-посилання і автоматично отримати синхронізовану послідовність акордів. Chordify частково вирішує цю задачу, але обмежений завантаженням аудіофайлів, не інтегрований в освітній процес і доступний лише в платній версії. Переважна більшість гітаристів-початківців продовжує вручну шукати табулатури, точність яких не завжди є задовільною [14].

Статичний контент і слабка персоналізація. Більшість застосунків пропонують фіксовану програму навчання, що не адаптується до рівня підготовки, темпу засвоєння матеріалу та музичних уподобань учня. Yousician та Fender Play пропонують певну структуру прогресії, проте вона є лінійною і не враховує індивідуальних прогалин у знаннях. Відсутність механізму персоналізованих рекомендацій призводить до того, що учень або рухається занадто повільно, повторюючи вже засвоєний матеріал, або пропускає важливі концепції, переходячи до складнішого рівня передчасно [8, 10].

Брак соціальної мотивації. Дослідження у сфері освітньої психології показують, що соціальна взаємодія та елементи гейміфікації суттєво підвищують мотивацію до навчання [15]. Проте жоден з аналізованих застосунків не реалізує повноцінної системи соціальних челленджів, де учні могли б змагатись між собою або мотивувати одне одного. Yousician має базову таблицю лідерів, проте відсутня механіка спільних завдань з дедлайнами та призами, що є ефективним інструментом утримання користувачів у довгостроковій перспективі.

Виявлені проблеми підтверджують доцільність розробки застосунку "Guitarly", який системно усуває кожне з описаних обмежень. AI-тьютор на основі RAG та LangGraph забезпечує інтелектуальну взаємодію з учнем; модуль ChordMini вирішує проблему розпізнавання акордів з YouTube-відео;

персоналізовані відповіді тьютора адаптуються до контексту запиту; система соціальних челленджів з дедлайнами та призами забезпечує довгострокову мотивацію.

1.2 Постановка завдання та цілей

Проведений аналіз предметної області та виявлені обмеження наявних рішень дають змогу чітко сформулювати завдання та цілі розробки програмної системи.

Метою розробки є проєктування та реалізація мобільного застосунку «Guitarly» для платформи Android, що поєднує персоналізованого AI-тьютора, автоматичне розпізнавання акордів із YouTube-відео та набір практичних інструментів музиканта (бібліотека акордів, тюнер, метроном) у межах єдиного рішення.

Для досягнення поставленої мети необхідно вирішити такі завдання: проаналізувати предметну область і наявні рішення для навчання гри на гітарі; визначити функціональні та нефункціональні вимоги до системи; спроєктувати клієнт-серверну архітектуру застосунку; реалізувати мобільний клієнт на Android із застосуванням принципів Clean Architecture; розробити серверну частину з AI-тьютором на основі RAG та оркестрації LangGraph; інтегрувати модуль автоматичного розпізнавання акордів; провести тестування та верифікацію системи.

Об'єктом дослідження є процес навчання гри на гітарі з використанням мобільних технологій та засобів штучного інтелекту.

Предметом дослідження є методи та засоби проєктування і реалізації мобільного застосунку з інтеграцією AI-тьютора, автоматичного розпізнавання акордів та допоміжних інструментів музиканта.

1.3 Пошук акторів та варіантів використання

Моделювання варіантів використання (Use Case) дозволяє описати взаємодію між акторами системи та функціями, які вона надає. Побудова такої моделі є першим кроком об'єктно-орієнтованого аналізу: вона переводить сформульовані вимоги у формалізований перелік сценаріїв, що згодом стають основою для проєктування архітектури та визначення меж системи. Виявлення акторів і прецедентів виконувалося на основі аналізу предметної області та функціональних вимог до застосунку для кожної вимоги визначалося, хто ініціює відповідну дію та який результат він очікує отримати. Для застосунку "Guitarly" виділено двох зовнішніх акторів та два системних сервіси, що беруть участь у реалізації основних сценаріїв.

Актори системи. Актором вважається будь-яка зовнішня сутність (людина або інша система), що взаємодіє із застосунком для досягнення певної мети. Основним (первинним) актором є Користувач – зареєстрований учень, який взаємодіє із застосунком через інтерфейс Android та ініціює переважну більшість сценаріїв: навчання, розпізнавання акордів, налаштування інструмента, участь у челенджах. Незареєстрований відвідувач (Гість) є окремим випадком актора з обмеженими правами він має доступ лише до екранів реєстрації та входу, а після авторизації успадковує всі можливості зареєстрованого користувача. Системними (вторинними) акторами є AI-сервіс (FastAPI-бекенд з RAG-тьютором та оркестрацією LangGraph), який генерує відповіді на запити щодо теорії гри, та ChordMini-сервіс, що виконує аналіз аудіо YouTube-відео й повертає синхронізовану послідовність акордів. Вторинні актори не ініціюють сценарії самостійно, а залучаються системою у відповідь на дії користувача.

Варіанти використання. На основі визначених акторів та функціональних вимог сформовано перелік ключових прецедентів, що охоплюють усі основні модулі застосунку: взаємодію з AI-тьютором, розпізнавання акордів у YouTube-відео, роботу з бібліотекою акордів, використання метронома та тюнера, участь у системі челенджів, а також авторизацію й керування профілем. Між окремими

прецедентами встановлено відношення включення (include) та розширення (extend): наприклад, сценарій розпізнавання акордів обов'язково включає отримання аудіодоріжки та її ML-аналіз, а перегляд альтернативних аплікатур розширює базовий сценарій перегляду діаграми акорду.

Повна UML-діаграма варіантів використання системи наведена в додатку А. Нижче описано ключові прецеденти, що відображають основні сценарії взаємодії користувача із системою.

1.4 Опис ключових варіантів використання

На основі проведеного аналізу предметної області та виявлених проблем визначено акторів системи та ключові варіанти використання застосунку «Guitarly», які надалі стають основою для формулювання функціональних вимог. Варіант використання описує завершений сценарій взаємодії актора із системою, що приводить до досягнення значущого для нього результату.

У системі виділено двох основних акторів-користувачів. Гість – неавторизований відвідувач, якому доступні функції навчання та інструментів без збереження персональних даних (бібліотека акордів, метроном, тюнер, реєстрація та вхід). Зареєстрований користувач успадковує всі можливості гостя та додатково отримує доступ до персоналізованих функцій: AI-тьютора, розпізнавання акордів у відео, персональних колекцій, челленджів і профілю. Окрім них, у сценаріях беруть участь зовнішні актори-системи: LLM-сервіс (генерація відповідей тьютора у режимі стрімінгу), YouTube та ML-сервіс розпізнавання акордів, мікрофон пристрою (захоплення звуку для тюнера) і поштовий сервіс (надсилання листів під час реєстрації та відновлення пароля).

– Взаємодія з AI-тьютором (ФВ-1). Користувач відкриває чат тьютора, формулює запитання щодо теорії гри й надсилає його. Система звертається до LLM-сервісу та поступово відображає відповідь у міру надходження токенів, зберігаючи контекст попередніх повідомлень у межах сесії. За потреби тьютор пропонує користувачу інтерактивні дії – відкрити акорд, перейти до пісні чи

запустити метроном, – і вибір такої дії розширює базовий сценарій переходом до відповідної підсистеми.

– Розпізнавання акордів у YouTube-відео (ФВ-2). Користувач вводить посилання на відео, після чого система отримує аудіодоріжку та передає її ML-сервісу для аналізу. У результаті формується послідовність акордів, яку застосунок відтворює синхронно з відео, показуючи графічні аплікатури. Додатково користувач може транспонувати отриману послідовність у зручну тональність.

– Робота з бібліотекою акордів (ФВ-3). Гість переглядає каталог акордів і відкриває діаграму аплікатури з позиціями пальців; за наявності альтернативних варіантів взяття вони відображаються як розширення базового сценарію. Зареєстрований користувач додатково додає акорди до персональних колекцій та позначає їх як обрані.

– Використання метронома (ФВ-4). Користувач запускає метроном і налаштовує темп (BPM), розмір такту й акценти; задані параметри зберігаються між сесіями. Окремий сценарій передбачає запуск метронома безпосередньо за рекомендацією AI-тьютора із вже підставленими параметрами.

– Налаштування гітари тюнером (ФВ-5). Користувач обирає тип строю (стандартний, Drop D, Open G, DADGAD тощо) та починає налаштування. Система захоплює звук через мікрофон, визначає висоту тону й показує візуальний індикатор відхилення від цільової ноти.

– Участь у челленджах (ФВ-6). Користувач переглядає список активних челленджів і відкриває деталі обраного – опис, правила, хештеги, дедлайн і приз. Він приєднується до челенджу та надсилає роботу, а також відстежує кількість учасників і поданих робіт.

– Авторизація та робота з профілем (ФВ-7). Гість реєструється за електронною поштою й паролем або входить у систему; за потреби ініціює відновлення пароля, що супроводжується надсиланням листа на пошту. Авторизований користувач переглядає профіль із персональними даними, своїми колекціями акордів та активними челленджами.

1.5 Функціональні вимоги

На основі проведеного аналізу предметної області та виявлених проблем сформульовано функціональні вимоги до застосунку "Guitarly". Вимоги визначають перелік функцій, які система повинна надавати користувачу, і є основою для проєктування архітектури та розробки програмного забезпечення.

AI-тьютор з підтримкою стрімінгу (ФВ-1). Система повинна надавати користувачу можливість ставити запитання щодо теорії гітарної гри та отримувати відповіді від AI-тьютора в режимі реального часу через стрімінг токенів. Тьютор має підтримувати збереження контексту розмови в рамках сесії та пропонувати дії (відкрити акорд, пісню, запустити метроном) у вигляді інтерактивних елементів інтерфейсу.

Розпізнавання акордів у YouTube-відео (ФВ-2). Користувач повинен мати можливість ввести посилання на YouTube-відео та отримати автоматично розпізнану, синхронізовану з відтворенням послідовність акордів. Система має відображати акорди в реальному часі разом з їх графічними аплікатурами та підтримувати транспонування результату.

Бібліотека акордів з графічними діаграмами (ФВ-3). Застосунок повинен містити повну бібліотеку гітарних акордів із графічними діаграмами аплікатур, позиціями пальців та альтернативними варіантами взяття. Користувач повинен мати можливість додавати акорди до персональних колекцій та позначати їх як обрані.

Цифровий метроном (ФВ-4). Застосунок повинен надавати функцію цифрового метронома з налаштуванням темпу (BPM), розміру такту та акцентів. Налаштування метронома мають зберігатись між сесіями та бути доступними для запуску безпосередньо з рекомендації AI-тьютора.

Хроматичний тюнер (ФВ-5). Система повинна забезпечувати хроматичне налаштування гітари з визначенням висоти звуку через мікрофон пристрою. Тюнер має підтримувати стандартне та альтернативні налаштування (Drop D, Open G, DADGAD тощо) з візуальним індикатором відхилення від цільової ноти.

Система челленджів (ФВ-6). Застосунок повинен надавати можливість переглядати активні челленджі, брати в них участь та відстежувати кількість учасників і надісланих робіт. Кожен челлендж має містити опис, правила, хештеги, дедлайн та приз для переможця.

Авторизація та профіль користувача (ФВ-7). Система повинна підтримувати реєстрацію та вхід за допомогою електронної пошти й пароля, а також відновлення пароля. Профіль користувача має відображати його персональні дані, колекції акордів та активні челленджі.

1.6 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи, що не пов'язані безпосередньо з її функціями, але суттєво впливають на користувацький досвід, безпеку та підтримуваність застосунку. Для системи "Guitarly" виділено п'ять категорій нефункціональних вимог: продуктивність, безпека, надійність, зручність використання та масштабованість. Деталізований перелік наведено в таблиці 1.2.

Таблиця 1.2 – Нефункціональні вимоги до застосунку "Guitarly"

Категорія	**Нефункціональна вимога**	**Показник / критерій**
1	**2**	**3**
Продуктивність	Час отримання першого токену від AI-тьютора	< 2 секунди при нормальному навантаженні
Продуктивність	Час завантаження екрану застосунку	< 1 секунда на пристроях середнього класу
Продуктивність	Час завершення аналізу акордів YouTube-відео	< 60 секунд для відео тривалістю до 5 хвилин
Безпека	Автентифікація користувача	JWT-токени з обмеженим терміном дії; HTTPS для всіх запитів
Безпека	Зберігання паролів	Хешування bcrypt на стороні сервера
Надійність	Обробка помилок SSE-стрімінгу	Автоматичне відновлення з'єднання; повідомлення користувача

Продовження таблиці 1.2

1	2	3
Зручність	Відповідність UI-гайдлайнам	Material Design 3; підтримка темної та світлої теми
Зручність	Підтримка Android-версій	Android 8.0 (API 26) та вище
Масштабованість	Розгортання серверної частини	Docker-контейнеризація; горизонтальне масштабування
Підтримуваність	Архітектура кодової бази	Clean Architecture; модульна структура; покриття тестами

Продуктивність є критичною характеристикою для застосунків з AI-компонентами. SSE-стрімінг відповідей тьютора дозволяє уникнути тривалого очікування: перший токен має з'явитись на екрані протягом двох секунд, а подальші токени надходять безперервно до завершення відповіді. Це забезпечує відчуття живої розмови замість очікування завантаження. Завантаження екранів застосунку реалізовано через асинхронні операції Kotlin Coroutines та Flow, що унеможливлює блокування UI-потоку.

Безпека забезпечується на рівні транспортного протоколу та автентифікації. Усі запити між Android-клієнтом та серверною частиною здійснюються виключно через HTTPS. Для авторизації використовуються JWT-токени з обмеженим терміном дії, що автоматично оновлюються через механізм refresh-токенів. Паролі користувачів зберігаються виключно у хешованому вигляді з використанням алгоритму bcrypt на стороні сервера [25].

Надійність SSE-з'єднання забезпечується механізмом обробки помилок на стороні Android-клієнта: у разі розриву з'єднання під час стрімінгу система відображає повідомлення про помилку та надає можливість повторити запит. Усі мережеві операції виконуються з таймаутами для запобігання зависанню інтерфейсу у разі недоступності сервера.

Зручність використання реалізована через дотримання гайдлайнів Material Design 3 від Google [7]. Застосунок підтримує системну темну та світлу теми, адаптивні розміри шрифтів та коректно відображається на пристроях з різними розмірами екрану. Мінімальна підтримувана версія Android 8.0 (API рівень 26), що охоплює понад 95% активних Android-пристроїв станом на 2024 рік [16].

Масштабованість та підтримуваність забезпечуються на архітектурному рівні. Серверна частина розгортається у Docker-контейнерах, що спрощує горизонтальне масштабування при зростанні навантаження. Android-застосунок побудовано на принципах Clean Architecture з чітким розділенням на domain-, data- та presentation-шари, що полегшує додавання нових feature-модулів без модифікації існуючого коду.

1.7 Вибір та обґрунтування технологічного стеку

Вибір технологій є одним із ключових архітектурних рішень, що визначають якість, підтримуваність та масштабованість системи. При формуванні стеку для "Guitarly" враховувались зрілість технологій, наявність активної спільноти, відповідність вимогам проєкту та досвід команди розробки. Технологічний стек поділено на три групи: мобільний застосунок, серверна частина та компоненти штучного інтелекту.

1.7.1 Мобільний застосунок (Android)

Android-застосунок розроблено мовою Kotlin (рис. 1.6) офіційною рекомендованою мовою розробки для платформи Android від Google [17]. Порівняно з Java, Kotlin забезпечує значно лаконічніший синтаксис, вбудовану підтримку null-safety, корутини для асинхронного програмування та повну сумісність з існуючими Java-бібліотеками. Kotlin Coroutines та Flow є основним механізмом реактивного програмування в застосунку: Flow використовується для стрімінгу SSE-подій від сервера та для спостереження за змінами локальної бази даних.

Рисунок 1.6 – Логотип Kotlin

Для побудови інтерфейсу користувача обрано Jetpack Compose (рис. 1.7) – сучасний декларативний UI-фреймворк від Google, який замінює традиційний XML-підхід. Compose дозволяє описувати UI як функції від стану, що значно спрощує тестування та усуває цілий клас помилок, пов'язаних із синхронізацією стану між View та ViewModel. Застосунок повністю побудований на Compose без використання XML-розмітки [18].

Рисунок 1.7 – Логотип Jetpack Compose

Ін'єкція залежностей реалізована через Hilt (рис. 1.8) бібліотеку, що базується на Dagger 2 та офіційно рекомендована Google для Android-проєктів. Hilt автоматично генерує код для створення та передачі залежностей між модулями, що суттєво знижує кількість шаблонного коду. Локальне збереження даних реалізовано через Room (рис. 1.8) ORM-бібліотеку поверх SQLite, що забезпечує типобезпечні SQL-запити та підтримку Kotlin Flow для реактивного читання даних. Мережева взаємодія з сервером виконується через Retrofit 3.0 з OkHttp 5.1; для SSE-

стрімінгу використовується анотація @Streaming з послідовним читанням байтів з ResponseBody. Повний перелік бібліотек Android-застосунку наведено в таблиці 1.3.

Рисунок 1.8 – Логотип Dagger Hilt та Room

Таблиця 1.3 – Технологічний стек Android-застосунку

Технологія	Версія	Призначення
Kotlin	2.3.20	Основна мова розробки застосунку
Jetpack Compose	BOM 2026.03	Декларативний UI-фреймворк
Hilt	2.59.2	Ін'єкція залежностей (DI)
Room	2.8.4	Локальна база даних (ORM)
Retrofit + OkHttp	3.0 / 5.1	HTTP-клієнт та SSE-стрімінг
Kotlin Coroutines	1.10.2	Асинхронне програмування та Flow
Navigation Compose	2.9.7	Навігація між екранами
Coil	3.3.0	Завантаження зображень
DataStore	1.2.1	Зберігання налаштувань
Android YouTube Player	13.0.0	Вбудований YouTube-плеєр

1.7.2 Серверна частина

Серверна частина системи побудована на FastAPI (рис. 1.9) – сучасному Python-фреймворку для розробки REST API. FastAPI обрано за нативну підтримку асинхронного виконання (async/await), автоматичну генерацію OpenAPI-документації, інтеграцію з Pydantic для валідації даних та вбудовану підтримку Server-Sent Events через StreamingResponse. FastAPI демонструє одну з найвищих продуктивностей серед Python-фреймворків завдяки використанню ASGI-сервера Uvicorn [19].

Рисунок 1.9 – Логотип FastAPI

Для роботи з реляційною базою даних використовується SQLModel бібліотека, що поєднує Pydantic-моделі з SQLAlchemy, дозволяючи використовувати одні й ті самі класи як для валідації API-запитів, так і для ORM-операцій з базою даних. Як СУБД обрано PostgreSQL 16, надійну реляційну базу даних з широкою підтримкою та розвиненою екосистемою. Розгортання системи виконується через Docker Compose, що забезпечує відтворюваність середовища та спрощує налаштування залежних сервісів. Технологічний стек серверної частини наведено в таблиці 1.4.

Таблиця 1.4 – Технологічний стек серверної частини

Технологія	**Версія**	**Призначення**
FastAPI	≥ 0.114	Веб-фреймворк для REST API та SSE
SQLModel	≥ 0.0.21	ORM-шар на основі Pydantic + SQLAlchemy
Pydantic	> 2.0	Валідація даних та схеми
PostgreSQL	16	Реляційна база даних
Docker	24+	Контейнеризація та розгортання

1.7.3 Компоненти штучного інтелекту

Для реалізації AI-тьютора обрано фреймворк LangChain (рис. 1.10) як базовий інструмент побудови RAG-пайплайнів. LangChain надає уніфікований інтерфейс для роботи з різними векторними сховищами, LLM-провайдерами та ланцюжками обробки документів, що дозволяє гнучко замінювати компоненти без переписування бізнес-логіки [20].

Рисунок 1.10 – Логотип LangChain

Оркестрацію агентів реалізовано через LangGraph – бібліотеку для побудови мультиагентних систем у вигляді графів станів (StateGraph). LangGraph дозволяє визначати вузли обробки (retrieve, grade_documents, generate, tool_use тощо) та умовні переходи між ними, що забезпечує гнучкість логіки Self-RAG: система здатна повторно виконати пошук у разі низької релевантності знайдених документів [21].

Як векторне сховище для зберігання embeddings навчальних матеріалів обрано Qdrant (рис. 1.11) – спеціалізовану векторну базу даних з підтримкою фільтрації за метаданими та горизонтального масштабування. Qdrant розгортається у Docker-контейнері та взаємодіє з бекендом через офіційний Python SDK. Для генерації embeddings використовується Ollama з локальною моделлю через інтеграцію langchain-ollama. LLM-генерацію відповідей виконує Groq API, який є хмарним провайдером з доступом до моделей Llama та Mixtral, що забезпечує низьку затримку завдяки апаратному прискоренню на LPU [22].

Рисунок 1.11 – Логотип Qdrant

Для автоматичного розпізнавання акордів використовується ChordMini, що є спеціалізованою моделлю глибинного навчання, що базується на архітектурі

Chord-CNN-LSTM. Модель приймає на вхід аудіодоріжку у форматі WAV та повертає часову послідовність акордів з мітками початку та кінця кожного акорду. ChordMini розгортається як окремий мікросервіс і взаємодіє з основним бекендом через внутрішній HTTP API з токен-автентифікацією. Повний перелік AI-компонентів наведено в таблиці 1.5.

Таблиця 1.5 – Технологічний стек компонентів штучного інтелекту

Технологія	Версія	Призначення
LangChain	≥ 1.0	Базовий фреймворк для RAG-пайплайну
LangGraph	≥ 1.0	Оркестрація агентів через StateGraph
langchain-qdrant	≥ 1.0	Інтеграція LangChain з Qdrant
langchain-ollama	≥ 1.0	Ollama-embeddings для векторизації
Qdrant	≥ 1.10	Векторна база даних для RAG
ChordMini	custom	Модель розпізнавання акордів з аудіо
Groq API	–	LLM-провайдер для генерації відповідей

1.8 Висновки до розділу 1

У першому розділі проведено комплексний аналіз предметної області навчання гри на гітарі за допомогою мобільних технологій. Огляд п'яти провідних конкурентних рішень виявив системні недоліки наявних застосунків: відсутність персоналізованого AI-тьютора, неможливість автоматичного розпізнавання акордів з YouTube-відео, статичний характер контенту та брак соціальних механізмів мотивації.

На основі виявлених проблем сформульовано сім функціональних вимог до системи "Guitarly" та десять нефункціональних вимог, що охоплюють продуктивність, безпеку, надійність, зручність використання та масштабованість. Побудовано модель варіантів використання з шістьма ключовими прецедентами для основного актора.

Обґрунтовано вибір технологічного стеку: Kotlin та Jetpack Compose для Android-клієнта, FastAPI та PostgreSQL для серверної частини, LangChain, LangGraph та Qdrant для реалізації RAG-пайплайну, ChordMini для розпізнавання

акордів. Отримані результати є основою для проєктування та розробки системи, що описується в наступному розділі.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

Другий розділ присвячено детальному проєктуванню та реалізації системи "Guitarly". Описано загальну архітектуру системи, мультимодульну структуру Android-застосунку, принципи Clean Architecture, механізм SSE-стрімінгу, RAG-пайплайн з LangGraph-оркестрацією, модуль розпізнавання акордів ChordMini та всі ключові feature-модулі застосунку.

2.1 Вибір процесу розробки

Розробка програмної системи «Guitarly» велася за ітеративно-інкрементною моделлю життєвого циклу, що поєднує гнучкість Agile-підходу з покроковим нарощуванням функціональності. Робота поділялася на короткі ітерації, у межах кожної з яких реалізовувався окремий feature-модуль (наприклад, бібліотека акордів, тюнер, AI-тьютор) з подальшим тестуванням та інтеграцією.

Для керування вихідним кодом використано систему контролю версій Git. Кожна нова функціональність розроблялася в окремій гілці, що після рев’ю та проходження автоматичних перевірок зливалася в основну гілку. Такий підхід забезпечив ізоляцію змін, можливість паралельної розробки клієнтської та серверної частин і прозору історію проєкту.

Складання, перевірку та доставку коду автоматизовано засобами безперервної інтеграції та доставки (CI/CD): для Android-клієнта застосовується система збірки Gradle із Convention Plugins, для серверної частини це контейнеризація на основі Docker та Docker Compose. Автоматичні тести (JUnit для Android та pytest для бекенду) запускаються на кожній ітерації, що дає змогу виявляти регресії на ранніх етапах.

Обраний процес розробки безпосередньо вплинув на архітектурні рішення, які детально розглянуто в наступних підрозділах.

2.2 Проектування архітектури системи

Архітектура системи «Guitarly» проєктувалася з урахуванням вимог масштабованості, тестованості та незалежності компонентів. У цьому підрозділі розглянуто загальну структуру системи, організацію мобільного клієнта за принципами Clean Architecture, мультимодульну будову проєкту та серверну інфраструктуру.

2.2.1 Загальна архітектура системи

Система "Guitarly" побудована за клієнт-серверною архітектурою і складається з трьох основних компонентів: Android-клієнта, основного FastAPI-бекенду та ChordMini-мікросервісу. Взаємодія між клієнтом і сервером здійснюється через захищене HTTPS-з'єднання з використанням JWT-автентифікації. Загальну архітектурну схему системи наведено на рисунку 2.1.

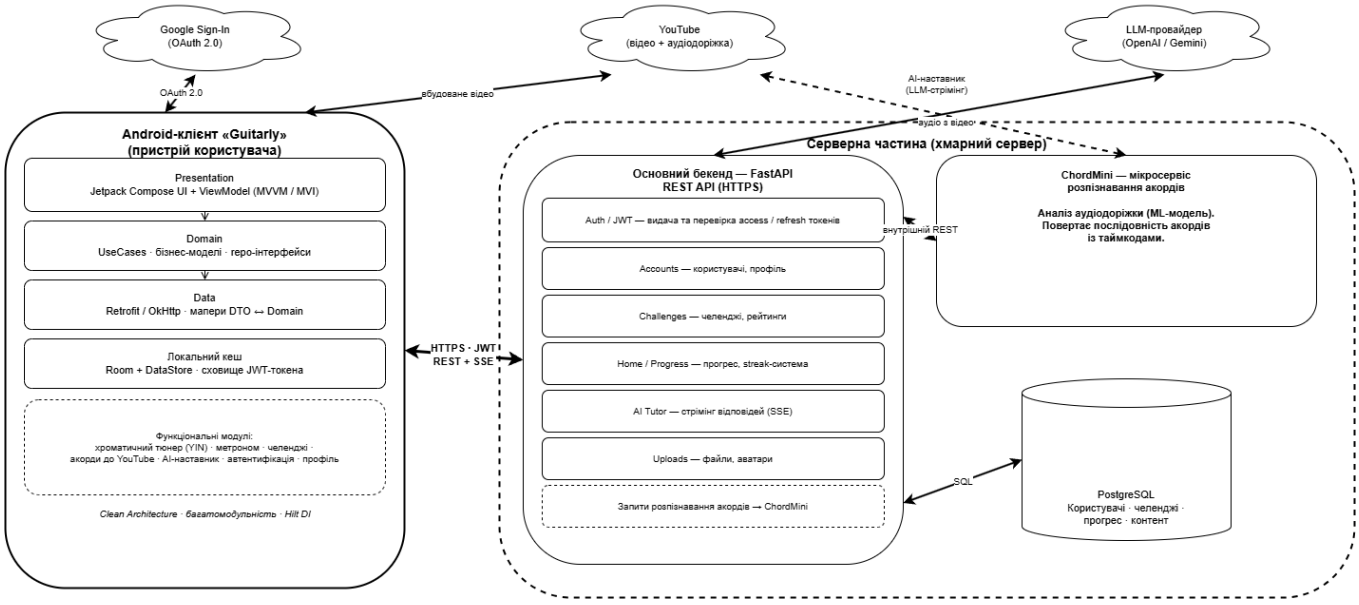


Рисунок 2.1 – Загальна архітектура системи "Guitarly"

Android-застосунок надсилає REST-запити до FastAPI-бекенду для всіх операцій, крім отримання відповідей AI-тьютора. Для стрімінгу токенів тьютора використовується протокол Server-Sent Events (SSE): клієнт відкриває

довготривале HTTP-з'єднання, через яке сервер надсилає потік подій у форматі text/event-stream. Це дозволяє відображати відповідь тьютора поступово, ще під час її генерації, без очікування повної відповіді.

FastAPI-бекенд є центральним компонентом системи: він обробляє запити автентифікації, управляє сесіями тьютора, зберігає дані в PostgreSQL та координує роботу AI-компонентів. При надходженні повідомлення до тьютора бекенд запускає LangGraph StateGraph, який виконує RAG-пошук у Qdrant, оцінює релевантність документів, викликає інструменти та генерує відповідь через Groq API, стрімлячи токени назад до клієнта через SSE.

ChordMini-мікросервіс є ізольованим Python-сервісом з HTTP API, що приймає шлях до аудіофайлу та повертає часову послідовність акордів. Основний бекенд взаємодіє з ним через внутрішню мережу Docker Compose, використовуючи токен-автентифікацію для захисту ендпоінту. Завдання аналізу акордів ставляться в асинхронну чергу, а клієнт опитує статус через polling.

Android-застосунок "Guitarly" побудований за принципами Clean Architecture та організований як мультимодульний Gradle-проєкт. Така структура забезпечує чітке розділення відповідальностей, прискорює інкрементальну збірку та спрощує паралельну розробку незалежних функціональних модулів.

2.2.2 Мультимодульна структура проекту

Проєкт складається з 42 Gradle-модулів, згрупованих у сім категорій: застосунок (app, app-demo), ядро (core), функціональні модулі (features), шар даних (data), з'єднувач (glue), навігація (navigation) та утиліти (utils). Граф залежностей між основними модулями наведено на рисунку 2.2. Повний перелік модулів описано в таблиці 2.1.

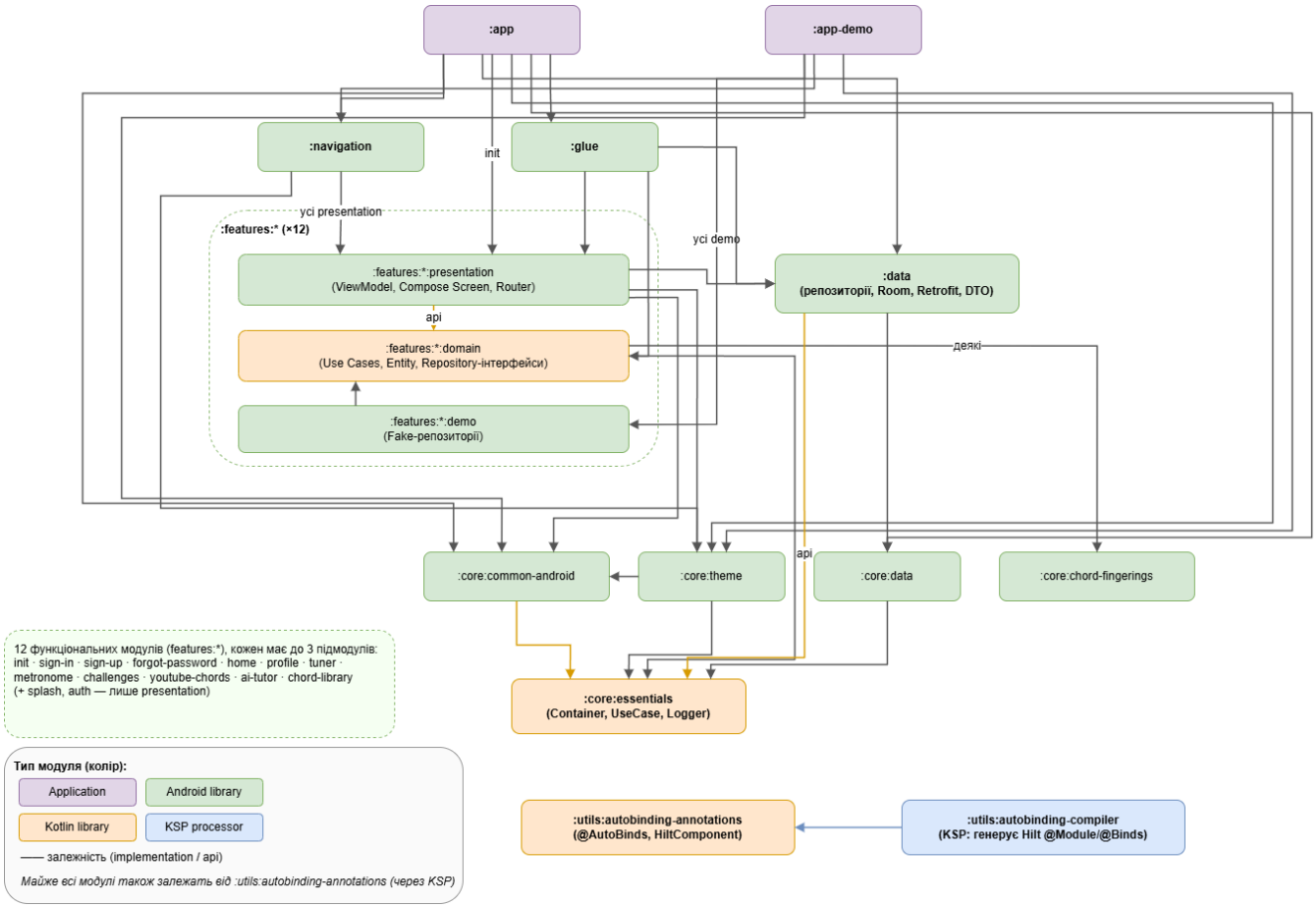


Рисунок 2.2 – Граф залежностей між модулями застосунку "Guitarly"

Таблиця 2.1 – Структура модулів застосунку "Guitarly"

Модуль	Тип	Призначення
1	2	3
:app	Application	Точка входу застосунку, Application-клас, ініціалізація Hilt
:app-demo	Application	Standalone-застосунок для запуску окремих feature у Compose Preview та на пристрої без сервера
:core:essentials	Kotlin lib	Базові абстракції: Container, UseCase, Logger, ExceptionHandler
:core:common-android	Android lib	StateViewModel, AndroidExceptionHandler, сповіщення, архітектурні базові класи
:core:theme	Android lib	Кольорова схема Material Design 3, типографіка, форми
:core:chord-fingerings	Android lib	Статичні дані аплікатур акордів (фрети, пальці, баррє)

Продовження таблиці 2.1

1	2	3
:core:data	Android lib	Спільні дані: AppDataStore, UserSettings, глобальний DataStore
:data	Android lib	Реалізації репозиторіїв, Room БД, Retrofit API, DTO-класи
:features:*:domain	Kotlin lib	Use Cases, Entity, Repository-інтерфейси (без Android-залежностей)
:features:*:presentation	Android lib	ViewModel, Compose Screen, Router-інтерфейс та його реалізація
:features:*:demo	Android lib	Fake-репозиторії для Compose Preview та app-demo
:glue	Android lib	Hilt-модулі: прив'язка domain-інтерфейсів до data-реалізацій
:navigation	Android lib	AppRoutes, AppNavGraph, AppNavHost, роутер-реалізації
:utils:autobinding-annotations	Kotlin lib	Анотація @AutoBinds та enum HiltComponent для кодогенерації
:utils:autobinding-compiler	KSP processor	KSP SymbolProcessor: генерує Hilt @Module/@Binds з @AutoBinds

Кожен feature-модуль поділяється на три субмодулі: domain, presentation та demo. Субмодуль domain є чистою Kotlin-бібліотекою без залежностей від Android SDK, що забезпечує максимальну тестованість бізнес-логіки. Presentation-субмодуль залежить від domain і містить Compose-екрани, ViewModel та Router. Demo-субмодуль надає fake-реалізації репозиторіїв для використання в Compose Preview та автономному запуску feature.

Модуль :app-demo є окремим Android-застосунком, що підключає demo-субмодулі всіх feature. Це дозволяє запускати будь-який екран у повністю ізольованому середовищі без реального сервера та бази даних безпосередньо на пристрої або в емуляторі. Такий підхід суттєво прискорює розробку та інспекцію UI: розробник може бачити живий екран з реалістичними тестовими даними, не запускаючи основний застосунок та серверну інфраструктуру.

Налаштування модулів уніфіковано через систему Convention Plugins, реалізовану в модулі :build-logic. Замість дублювання конфігурацій Gradle кожен модуль застосовує один із преднастроєних плагінів: guitarly.android.feature для feature-модулів, guitarly.kotlin.domain для domain-бібліотек, guitarly.android.room

для модулів з Room тощо. Це знижує обсяг шаблонного коду та гарантує консистентність конфігурацій по всьому проєкту.

Кастомна кодогенерація (@AutoBinds). У модулях :utils:autobinding-annotations та :utils:autobinding-compiler реалізовано власний KSP-процесор символів (SymbolProcessor), що усуває потребу вручну писати шаблонні Hilt @Module/@Binds класи. Анотація @AutoBinds(installIn = HiltComponent.Singleton) розміщується на класі-реалізації репозиторію, після чого KSP під час збірки автоматично генерує відповідний Hilt-модуль через KotlinPoet. Наприклад, для класу AiTutorFeatureRepository, що реалізує інтерфейс TutorRepository, генерується модуль AiTutorFeatureRepositoryModule з абстрактним методом bindToTutorRepository. Це скорочує обсяг boilerplate-коду в :glue і унеможливлює помилки, пов'язані з ручним написанням binding-модулів.

2.2.3 Clean Architecture та шари застосунку

Кожен функціональний модуль побудований за трьома шарами Clean Architecture [23]: domain, data та presentation. Залежності спрямовані виключно всередину: presentation залежить від domain, data реалізує інтерфейси domain, але domain не залежить від жодного з них. Схему шарів наведено в додатку В.

Domain-шар містить бізнес-сутності (entity), інтерфейси репозиторіїв та Use Cases класи, що інкапсулюють одну конкретну бізнес-операцію. Наприклад, для модуля ai-tutor domain-шар визначає TutorRepository з методами createSession, sendMessage та getSessionMessages, а також три відповідні Use Cases: CreateTutorSessionUseCase, SendTutorMessageUseCase та GetTutorSessionMessagesUseCase. Use Cases реалізуються як окремі класи з єдиним методом invoke, що відповідає принципу єдиної відповідальності.

Data-шар зосереджений у спільному модулі :data та реалізує інтерфейси репозиторіїв, визначені в domain. Кожна група репозиторіїв містить Remote API-інтерфейс Retrofit, DTO-класи для серіалізації JSON, локальні Room Entity та DAO для кешування, а також клас DataRepositoryImpl, що координує роботу remote та

local джерел. Реалізації репозиторіїв позначені анотацією @AutoBinds, завдяки чому KSP автоматично генерує Hilt binding-модуль для кожної з них.

Presentation-шар побудований за шаблоном MVI (Model-View-Intent) (рис. 2.4) на основі власного базового класу StateViewModel<S>. Кожен ViewModel успадковує StateViewModel, що надає StateFlow<Container<S>> зі станами Loading, Success та Error. Compose-екран підписується на цей StateFlow через collectAsStateWithLifecycle() та відображає відповідний UI залежно від поточного стану. Навігаційні переходи делегуються окремому інтерфейсу Router, реалізація якого знаходиться в модулі :navigation, що дотримується принципу інверсії залежностей.

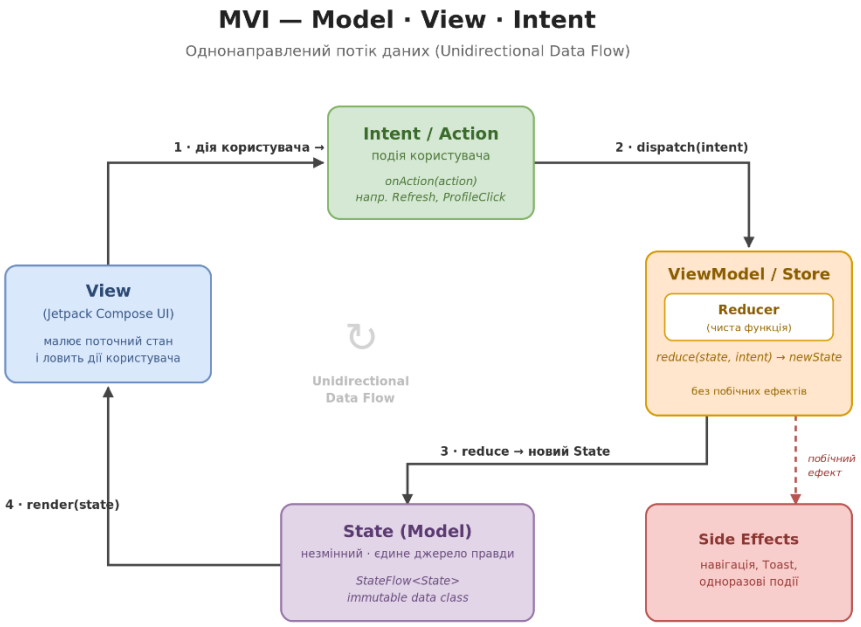


Рисунок 2.4 – MVI архітектура

2.2.4 Серверна інфраструктура

Серверна частина застосунку побудована на основі шаблону full-stack-fastapi-template і виконує роль посередника між Android-клієнтом та AI-компонентами. Основний стек включає FastAPI як ASGI-фреймворк, PostgreSQL як реляційну базу даних, SQLModel для ORM-взаємодії та Alembic для міграцій схеми. Celery-воркер

обробляє фонові задачі зокрема, асинхронне завантаження знань до векторного сховища. Уся інфраструктура упакована в Docker-контейнери та оркеструється через Docker Compose, що забезпечує відтворюваність середовища розгортання та горизонтальне масштабування окремих сервісів.

Маршрути API організовані у вигляді окремих роутерів FastAPI, які реєструються в головному застосунку через include_router(). Аутентифікація реалізована через JWT-токени: при вході користувач отримує access- та refresh-токени, які передаються у заголовку Authorization. Для захисту всіх ендпоінтів AI-тьютора використовується залежність CurrentUser, яка декодує токен та завантажує об'єкт авторизованого користувача з бази даних. Пряма розробка цієї інфраструктурної частини не входила до завдань даної роботи – вона слугувала готовою платформою для реалізації AI-компонентів, описаних у підрозділах 2.4 та 2.5.

2.3 Побудова схем бази даних

Для зберігання даних на стороні клієнта застосовується вбудована СУБД SQLite. Доступ до даних реалізовано через ORM-бібліотеку Room із пакету Android Jetpack, яка забезпечує типобезпечну роботу з базою даних через анотовані Kotlin-класи та інтерфейси DAO. Для зберігання легковагих налаштувань і токена автентифікації використовується Preferences DataStore.

Схема бази даних реалізована у вигляді єдиної бази HomeDatabase (версія 2), що логічно поділяється на три групи сутностей: профіль та активність користувача, кеш пісень з акордами і загальне сховище ключ-значення. Центральною сутністю є профіль користувача (home_profile), з яким пов’язані таблиці активності та вивчених пісень.

Група профілю та активності містить таблиці home_profile із кешованими даними облікового запису, home_activity для запису сесій практики та home_learned_song для відстеження вивчених пісень. Група пісень включає youtube_cached_song із серіалізованими даними треку та youtube_song_state зі

станом відтворення (каподастр, транспозиція, улюблене). Таблиця app_key_value є загальним JSON-сховищем для серіалізованих налаштувань метронома та програми. Токен автентифікації зберігається окремо у Preferences DataStore. Повна схема зв'язків між сутностями наведена на ER-діаграмі (рис. 2.5).

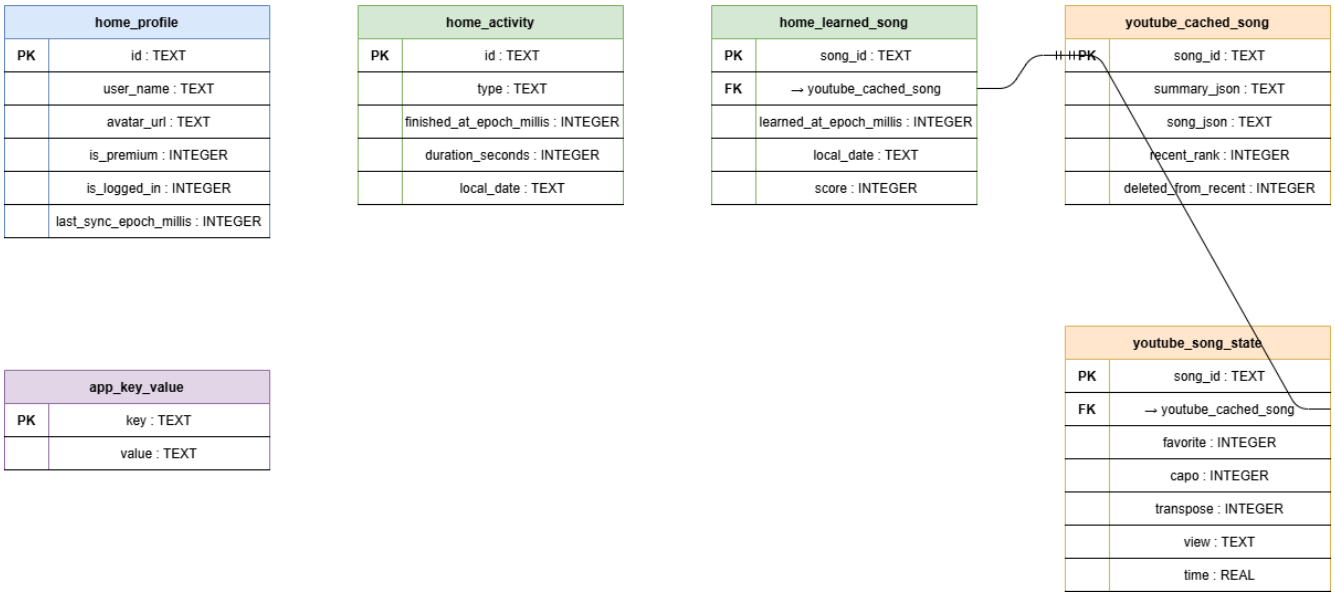


Рисунок 2.5 – ER-діаграма локальної бази даних застосунку «Guitarly»

Таблиця 2.2 – Сутності локальної бази даних застосунку «Guitarly»

Сутність	**Призначення**	**Ключові поля**
home_profile	Кешований профіль користувача	id, user_name, avatar_url, is_premium, last_sync_epoch_millis
home_activity	Журнал сесій практики	id, type, finished_at_epoch_millis, duration_seconds, local_date
home_learned_song	Вивчені пісні користувача	song_id, learned_at_epoch_millis, local_date, score
youtube_cached_song	Кеш даних пісні з YouTube	song_id, summary_json, song_json, recent_rank, deleted_from_recent
youtube_song_state	Стан відтворення пісні	song_id, favorite, capo, transpose, view, time
app_key_value	JSON-сховище налаштувань	key, value

2.4 UML-моделювання системи

Уніфікована мова моделювання (UML) є стандартизованим засобом візуального опису структури та поведінки програмних систем. Для проєктування системи «Guitarly» застосовано чотири взаємодоповнювальні види UML-діаграм: діаграми класів, що фіксують статичну структуру та зв'язки між компонентами; діаграми послідовності, які описують хронологію обміну повідомленнями між об'єктами; діаграми активності, що моделюють потоки керування й паралельні процеси; та діаграми станів, які визначають життєвий цикл ключових сутностей. Такий комплексний підхід дозволяє розглянути архітектуру системи з різних ракурсів, від статичної організації коду, до динаміки виконання сценаріїв. І забезпечує цілісне розуміння проєктних рішень ще до етапу реалізації. Моделювання виконано згідно з принципами чистої архітектури (Clean Architecture) з чітким розмежуванням доменного, презентаційного та інфраструктурного шарів.

2.4.1 Діаграми класів

Об’єктно-орієнтоване проєктування системи відображено за допомогою UML-діаграм класів, що описують статичну структуру основних компонентів, та діаграм послідовності, які ілюструють динаміку взаємодії об’єктів у часі.

Доменний шар Android-застосунку спроєктовано на основі принципу інверсії залежностей: для кожної операції визначено інтерфейс сценарію використання (UseCase), реалізація якого (UseCaseImpl) делегує роботу інтерфейсу репозиторію. Наприклад, у модулі бібліотеки акордів інтерфейс GetChordCatalogUseCase реалізується класом GetChordCatalogUseCaseImpl, що звертається до репозиторію акордів. Така структура забезпечує незалежність бізнес-логіки від конкретних джерел даних. Структуру класів доменного шару показано на рис. 2.6.

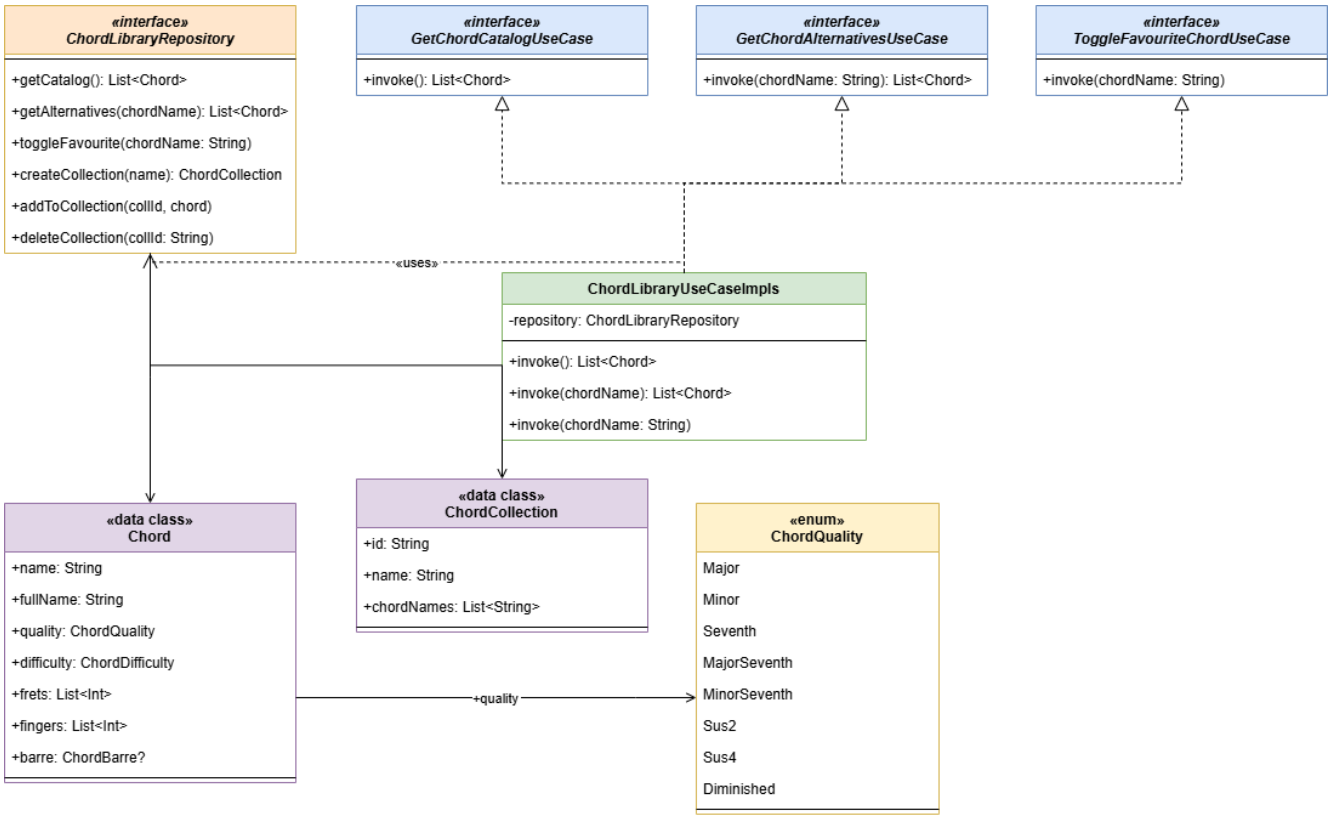


Рисунок 2.6 – UML-діаграма класів доменного шару Android-застосунку

Презентаційний шар побудовано за патерном MVI (Model-View-Intent): кожен екран описується незмінним станом (State), набором намірів користувача (Intent) та ViewModel, що перетворює наміри у нові стани. Класи станів та намірів є типобезпечними та інкапсулюють усю логіку відображення.

Доменний шар функції YouTube-акордів спроєктовано навколо інтерфейсу YoutubeChordsRepository, що визначає контракт для роботи з піснями та аналізом акордів. Кожній операції відповідає окремий UseCase-інтерфейс: StartAnalysisUseCase, GetSongUseCase, PatchSongStateUseCase та інші. Реалізації всіх UseCase-ів зосереджено в класі YoutubeChordsUseCaseImpls, який через ін'єкцію залежностей отримує посилання на репозиторій. Доменні сутності SongAnalysis, SongState та AnalysisJob є незмінними data class, що передають дані між шарами. Структуру класів наведено на рис. 2.7.

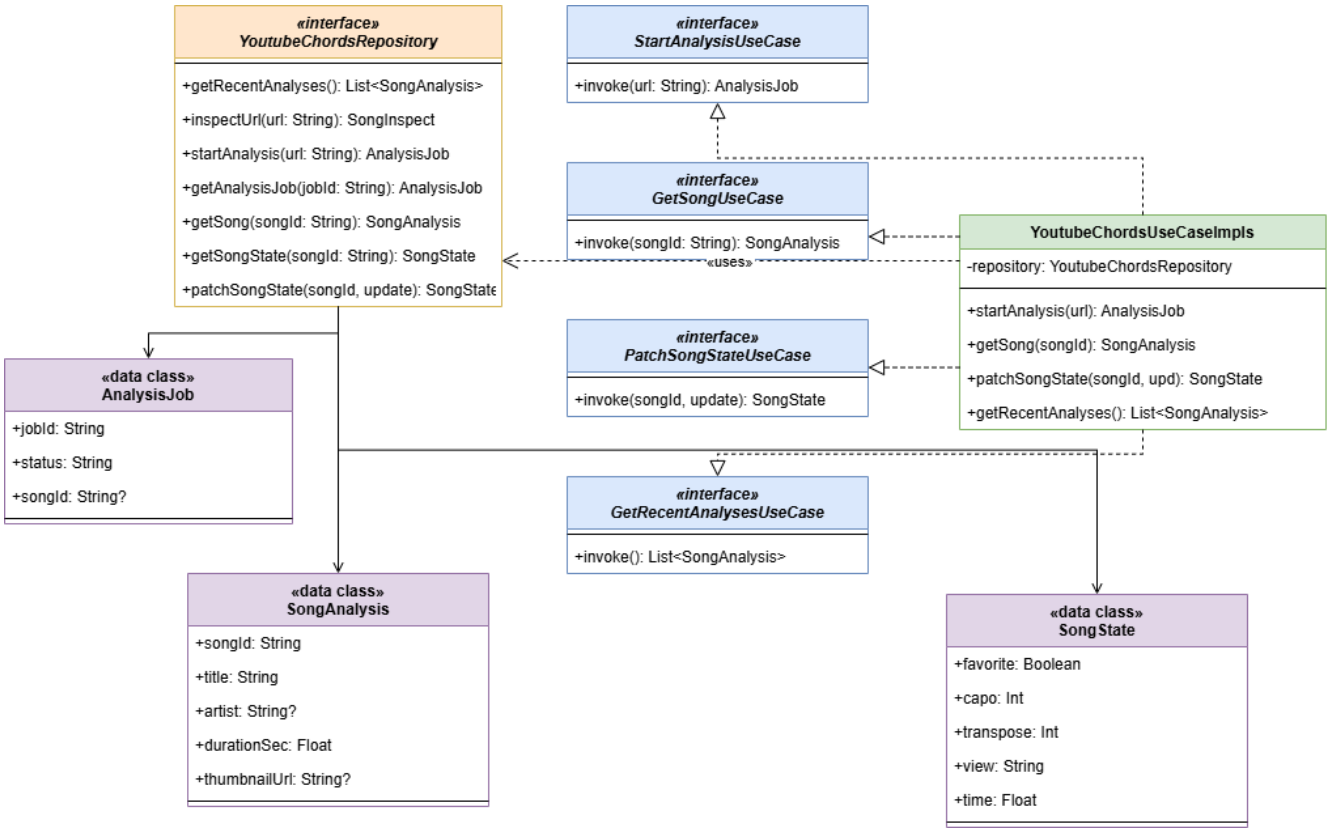


Рисунок 2.7 – UML-діаграма класів домену YouTube-акордів Android-застосунку

Презентаційний шар функції YouTube-акордів побудовано за патерном MVI (Model-View-Intent). Центральним класом є YoutubeViewModel, що реагує на наміри (Intent) з боку екрану, делегує бізнес-логіку відповідним UseCase-ам та публікує незмінний стан (State) у StateFlow. Екран YoutubeScreen підписується на StateFlow та перемальовується декларативно через Jetpack Compose. Класи стану містять усі дані, необхідні для відображення, включно з SongAnalysis, списком ChordLine та прапорцем завантаження. Структуру наведено на рис. 2.8.

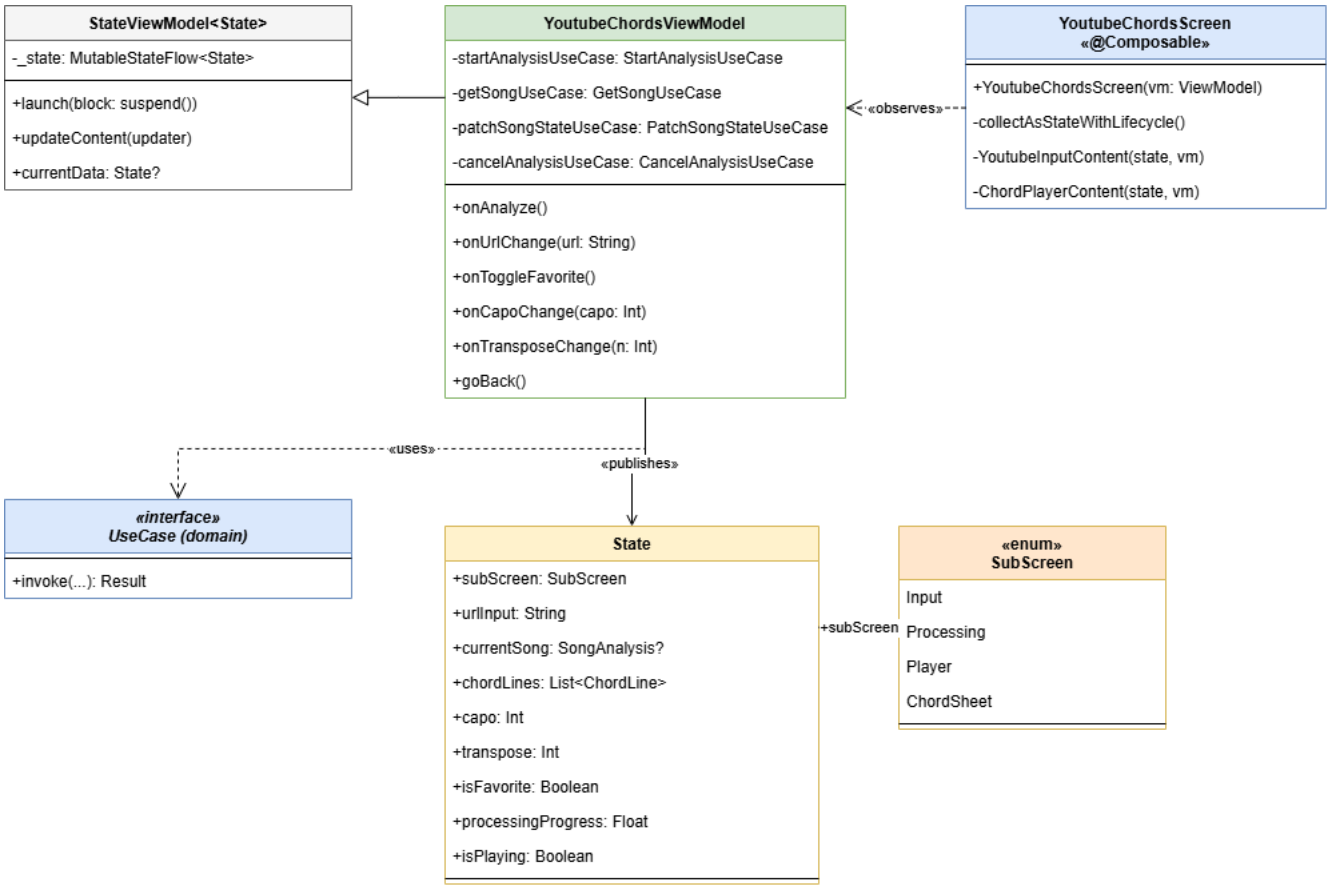


Рисунок 2.8 – UML-діаграма класів презентаційного шару YouTube-акордів (MVI)

2.4.2 Діаграми послідовності

Діаграми послідовності (Sequence Diagrams) відображають хронологічний порядок обміну повідомленнями між об’єктами системи під час виконання конкретного сценарію. Взаємодію компонентів з AI-тьютором у режимі стрімінгу детально описано на рисунку 2.8. Нижче наведено діаграми для двох додаткових ключових сценаріїв системи.

Діаграма послідовності процесу автентифікації (рисунок 2.9) охоплює сценарій реєстрації нового користувача та повторного входу. Новий користувач надсилає дані облікового запису через Android-клієнт до Auth Service. Auth Service перевіряє унікальність email у базі даних PostgreSQL, хешує пароль за алгоритмом bcrypt, зберігає запис і генерує пару JWT-токенів (access + refresh). При повторному вході клієнт надсилає credentials, отримує нові токени й зберігає їх у DataStore. У разі протермінування access-токена клієнт автоматично виконує рефреш через окремий endpoint без повторного введення пароля.

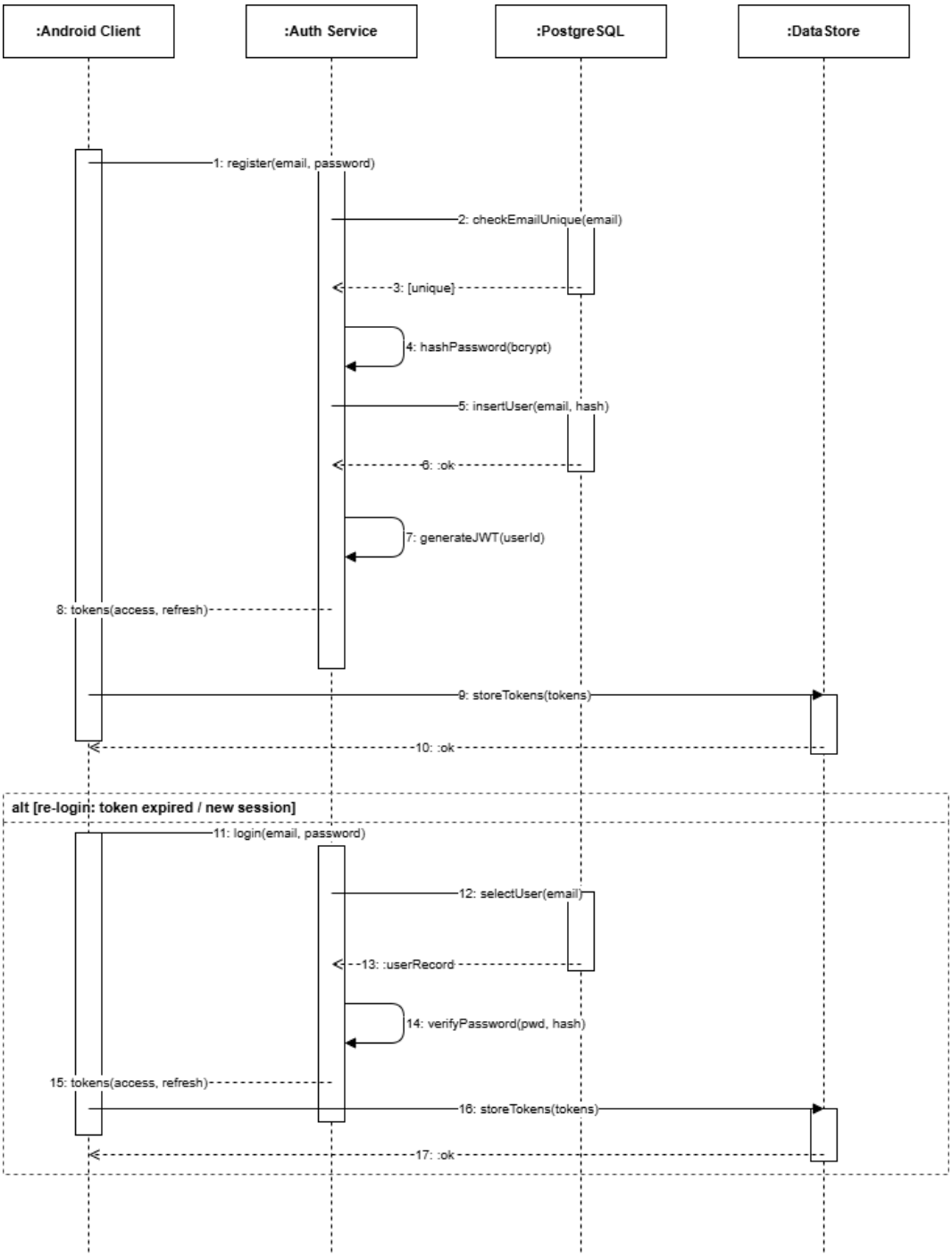


Рисунок 2.9 – Діаграма послідовності процесу автентифікації користувача

Діаграма послідовності для процесу розпізнавання акордів YouTube-відео (рисунок 2.10) ілюструє взаємодію між чотирма учасниками: Android-клієнтом, YouTube Data API, ChordMini-сервісом та серверною частиною Guitarly. Користувач вводить URL відео; клієнт витягує аудіодоріжку через YouTube Data API та надсилає буфер серверу. Сервер передає аудіо до ChordMini-мікросервісу, який повертає послідовність акордів із мітками часу. Сервер агрегує результати,

додає метадані (BPM, тональність) і повертає структурований JSON клієнту для відображення у касетному плеєрі.

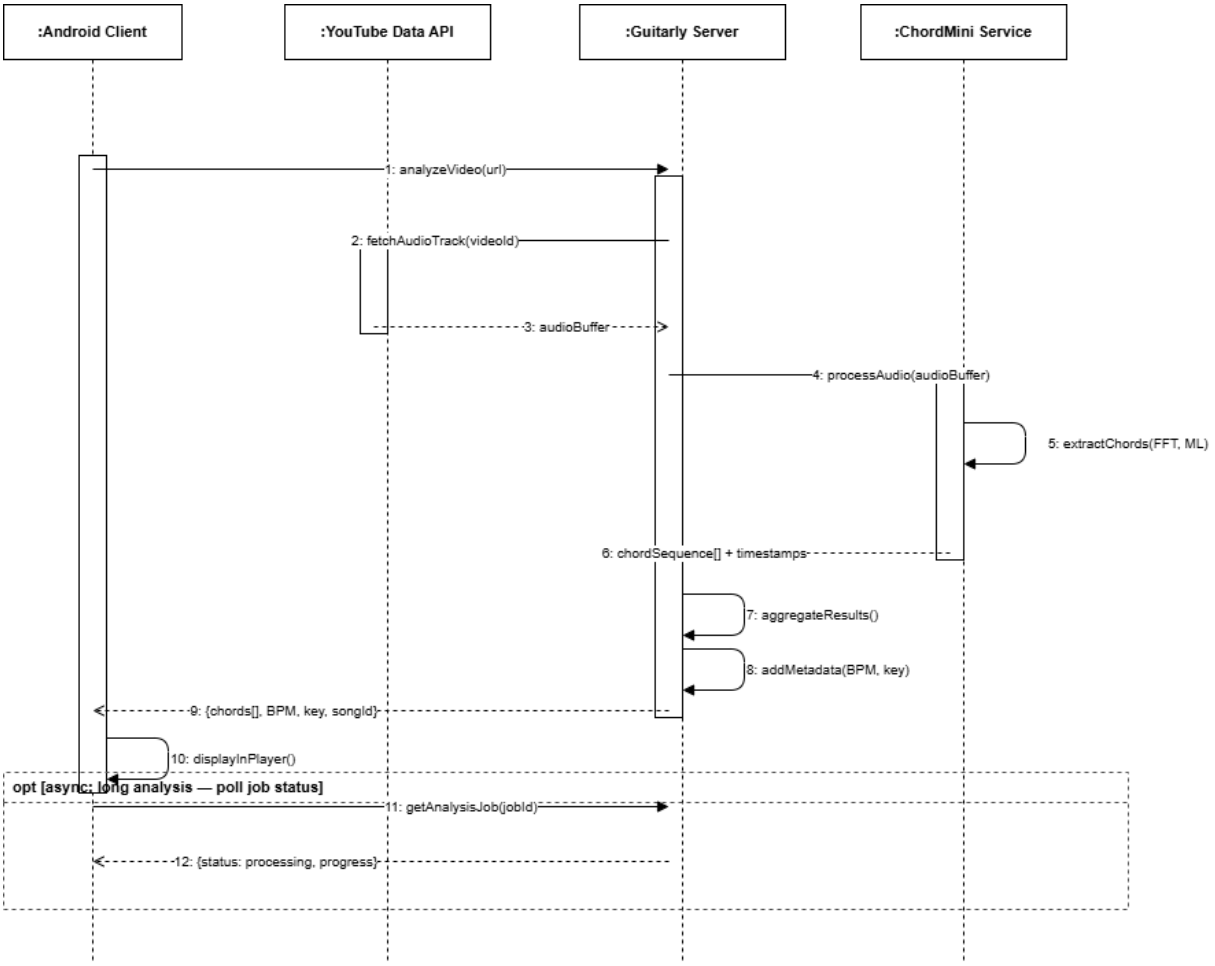


Рисунок 2.10 – Діаграма послідовності розпізнавання акордів YouTube-відео

2.4.3 Діаграми активності

Діаграми активності (Activity Diagrams) описують потік керування та паралельні процеси у системі «Guitarly». Вони є ефективним інструментом моделювання бізнес-процесів і складних алгоритмів, що включають умовні розгалуження та паралельні потоки обробки даних.

Діаграма активності процесу реєстрації та входу в систему (рисунок 2.11) демонструє розгалуження між двома сценаріями: нового та існуючого користувача. Для нового користувача потік охоплює введення даних, валідацію формату email та складності пароля, хешування за bcrypt, збереження в PostgreSQL і генерацію

JWT. Для існуючого користувача потік спрощується до перевірки credentials та видачі токенів. Після успішної автентифікації за обома гілками потік об’єднується: клієнт зберігає токени у DataStore та переходить на головний екран.

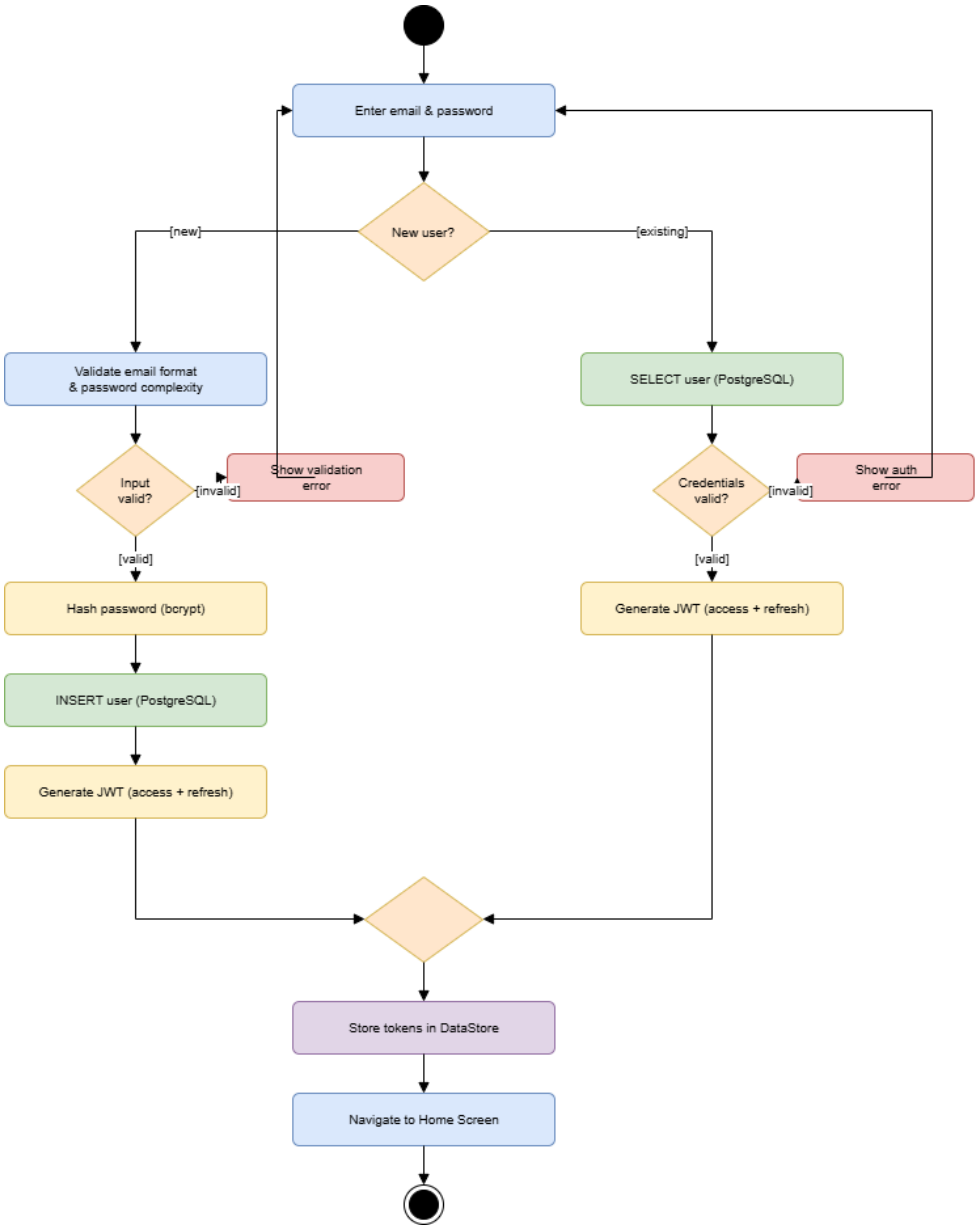


Рисунок 2.11 – Діаграма активності процесу реєстрації та входу в систему

Діаграма активності функції розпізнавання акордів (рисунок 2.12) моделює детальний алгоритм обробки запиту. Після отримання URL відео система паралельно витягує аудіо та метадані. Якщо тривалість відео перевищує 10 хвилин, активується автоматичне сегментування; інакше аудіо обробляється цілісно. ChordMini-сервіс виконує акордовий аналіз із визначенням часових міток.

Подальша обробка включає транспонування акордів у цільову тональність та побудову структурованої послідовності для відображення у касетному плеєрі.

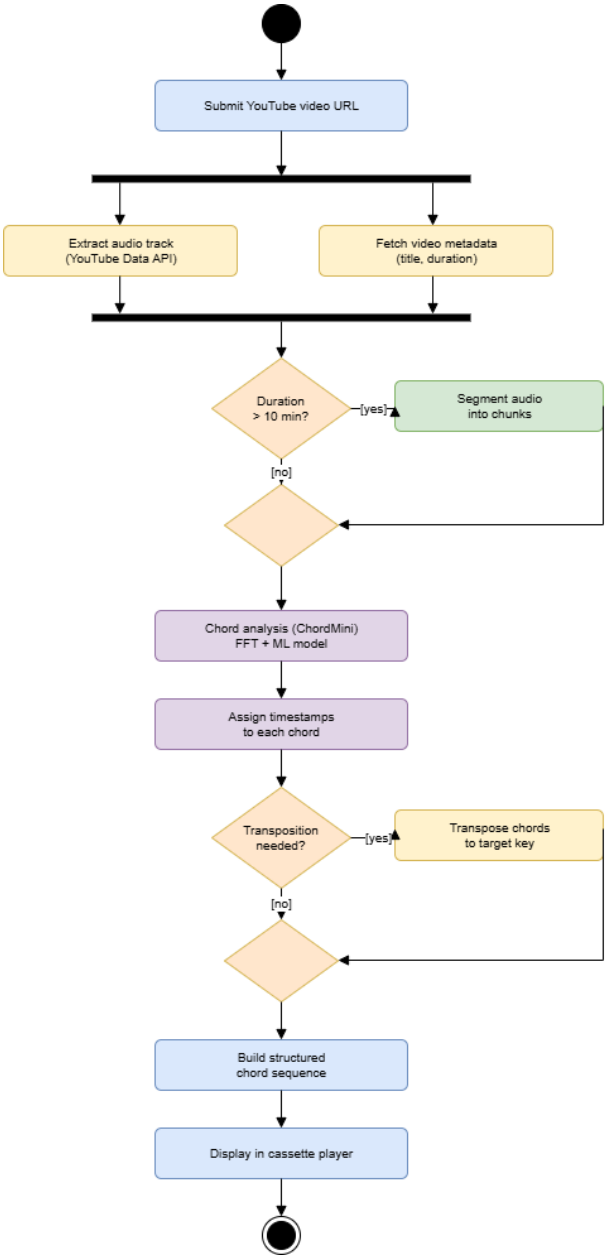


Рисунок 2.12 – Діаграма активності функції розпізнавання акордів YouTube-відео

2.4.4 Діаграми станів

Діаграми станів (State Diagrams) описують поведінку об’єктів системи у відповідь на зовнішні події, визначаючи допустимі стани та переходи між ними. У системі «Guitarly» діаграми станів застосовано для моделювання сесії AI-тьютора та життєвого циклу соціального челенджу.

Діаграма станів сесії взаємодії з AI-тьютором (рисунок 2.13) містить п’ять ключових станів. Початковий стан – Idle: клієнт підключений, але запит іще не надіслано. Після відправки повідомлення система переходить у Processing: класифікація теми та вибір стратегії генерації. За потреби звернення до бази знань активується стан Retrieving. Далі сесія переходить у Streaming: LLM генерує відповідь, що посимвольно надходить клієнту через SSE. Після отримання токена [DONE] система повертається у Idle. Стан Error є перехідним – будь-яка помилка призводить до повернення в Idle.

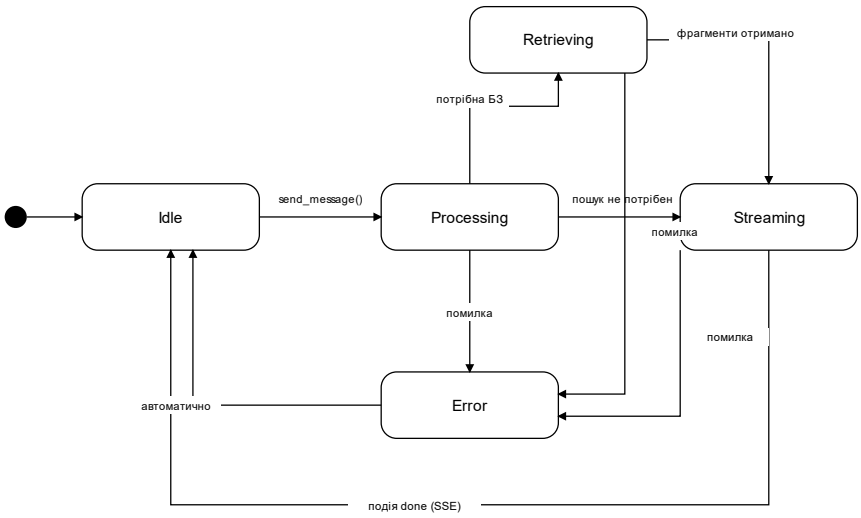


Рисунок 2.13 – Діаграма станів сесії взаємодії з AI-тьютором

Діаграма станів челенджу (рисунок 2.14) охоплює шість станів. Після публікації автором челендж перебуває у стані Active та доступний учасникам у стрічці. Користувач приймає виклик – його запис переходить у In Progress. Після завантаження виконання запис переходить у Submitted та очікує оцінювання. Голосування спільноти переводить запис у стан Evaluated з рейтингом. Автор може завершити змагання командою close, переводячи всі активні записи у Closed. Стан Expired передбачений для автоматичного закриття за спливом таймера (24 або 48 годин).

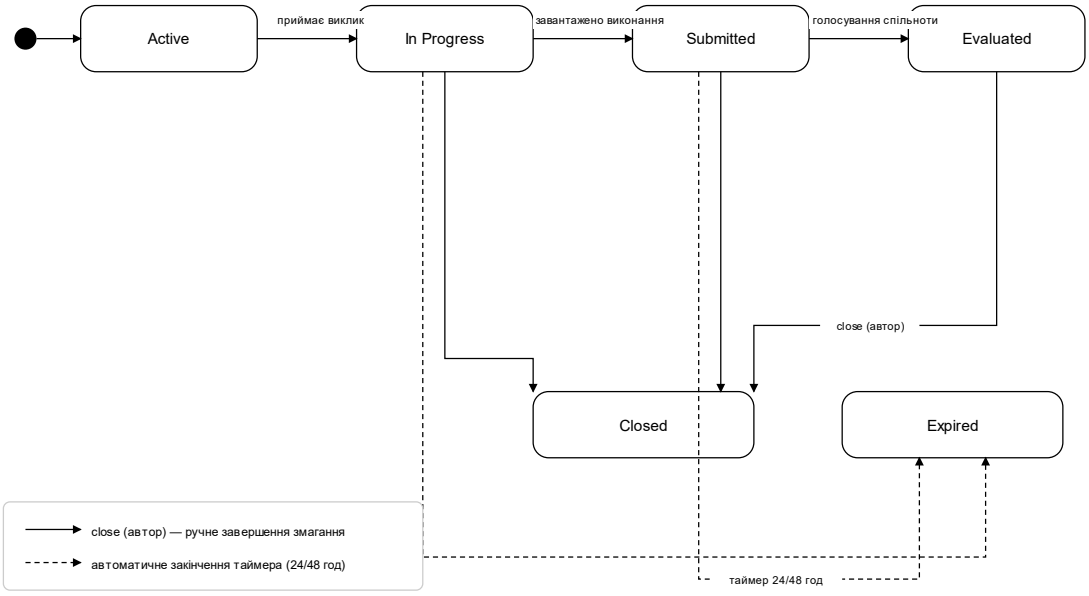


Рисунок 2.14 – Діаграма станів челенджу в системі «Guitarly»

2.5 Реалізація основних класів та методів

У цьому підрозділі описано реалізацію ключових класів та методів системи: серверного модуля AI-тьютора, функції розпізнавання акордів YouTube-відео та основних feature-модулів Android-застосунку.

2.5.1 Модуль AI-тьютора

Модуль AI-тьютора є центральним компонентом власної розробки в серверній частині застосунку та реалізований у пакеті app/tutor/. Архітектурно він являє собою агентний RAG-конвеєр (Retrieval-Augmented Generation), побудований на базі фреймворку LangGraph. Тьютор виконує дві принципово різні функції залежно від типу запиту: для питань про гітарну техніку, акорди та музичну теорію він витягує релевантні фрагменти з векторного сховища знань і генерує відповідь із цитатами; для інтерактивних команд показати діаграму акорду, знайти пісню, транспонувати або запустити метроном, він викликає відповідні інструменти та повертає структуровані результати у вигляді пропонованих дій.

Основний клас TutorGraph відповідає за компіляцію та виконання графу станів, а всі ключові компоненти, ланцюжки LangChain, вузли графу, інструменти та дескриптори стану організовані в окремих модулях: chains.py, graph.py, tools.py, prompts.py та state.py. Такий поділ забезпечує незалежну тестованість кожного рівня та спрощує подальше розширення функціоналу без внесення змін до головного класу. Загальну схему модуля AI-тьютора наведено на рисунку 2.15.

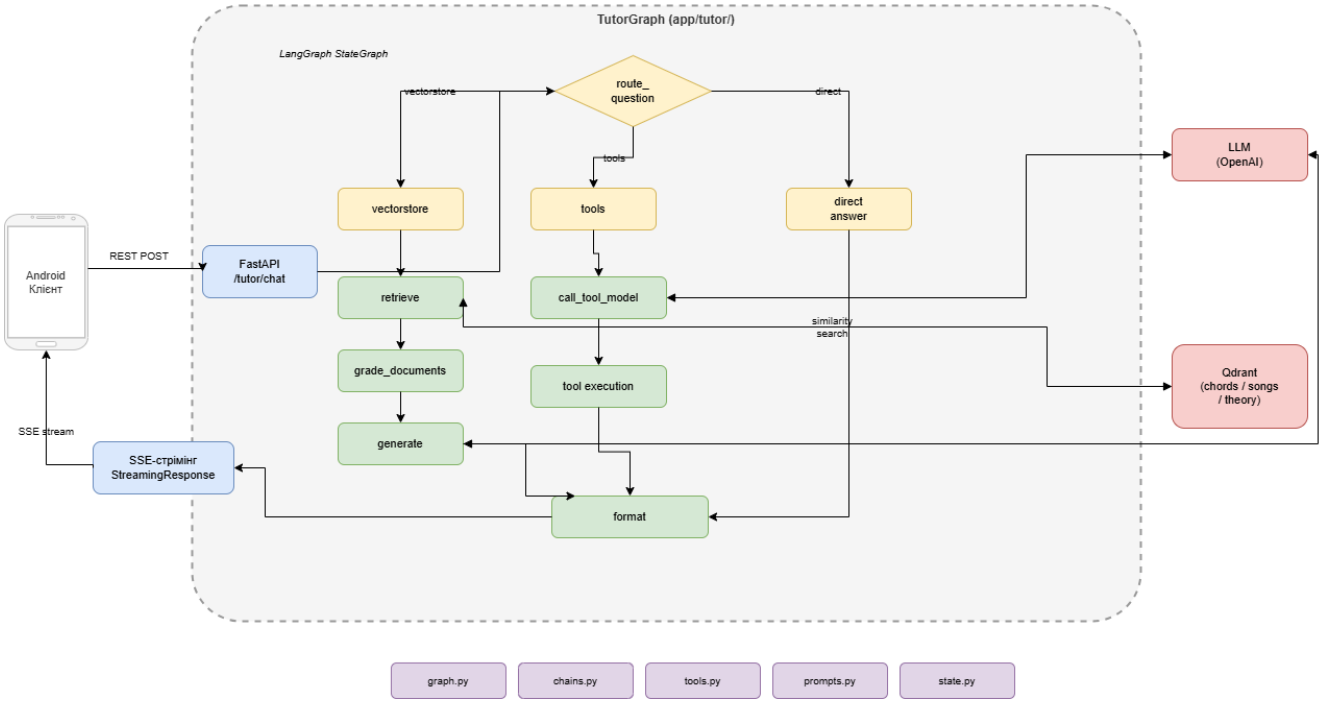


Рисунок 2.15 – Загальна схема модуля AI-тьютора

2.5.2 RAG-конвеєр та векторна база знань

Для зберігання та пошуку навчальних матеріалів використовується векторне сховище Qdrant – спеціалізована база даних для роботи з ембедингами. Знання організовані в три окремі колекції залежно від типу вмісту: guitar_kb_text зберігає фрагменти навчального тексту, guitar_kb_chords – фрагменти з великою кількістю назв акордів, guitar_kb_tabs – гітарні табулатури. Такий розподіл дозволяє при пошуку враховувати специфіку кожного типу матеріалу та підвищує точність релевантних результатів (рис. 2.17). Розподіл навчальних матеріалів між колекціями у Qdrant показано на рисунку 2.16.

Завантаження знань реалізоване у модулі rag.py через функцію ingest_document(), яка підтримує PDF- та TXT-формати. Документ завантажується через бібліотеку PdfReader або стандартне читання файлу, після чого розбивається на фрагменти за допомогою RecursiveCharacterTextSplitter зі збереженням семантичних меж: роздільниками служать заголовки Markdown, подвійні переноси рядка, речення та пробіли. Кожен фрагмент класифікується за регулярними виразами: якщо фрагмент містить рядки у форматі гітарної табулатури, він потрапляє до колекції guitar_kb_tabs; якщо містить три і більше назв акордів, то до колекції guitar_kb_chords, інакше – до колекції guitar_kb_text. Для кожного фрагменту обчислюється SHA-256 хеш, що запобігає дублюванню при повторному завантаженні тієї самої версії документа. Ідентифікатор фрагменту у Qdrant формується через UUID5 з урахуванням джерела, колекції, порядкового індексу та хешу вмісту.

Пошук у векторному сховищі виконується функцією retrieve(), яка одночасно звертається до всіх трьох колекцій через LangChain-інтерфейс QdrantVectorStore. Для кожної колекції використовується cosine similarity як метрика відстані; результати об'єднуються, сортуються за значенням схожості та дедублюються за хешем вмісту. Ембединги генеруються локально через Ollama з моделлю, що налаштовується через конфігурацію TUTOR_EMBEDDING_MODEL, а контекст токенізатора встановлений у 8192 токени для роботи з довгими музичними матеріалами.

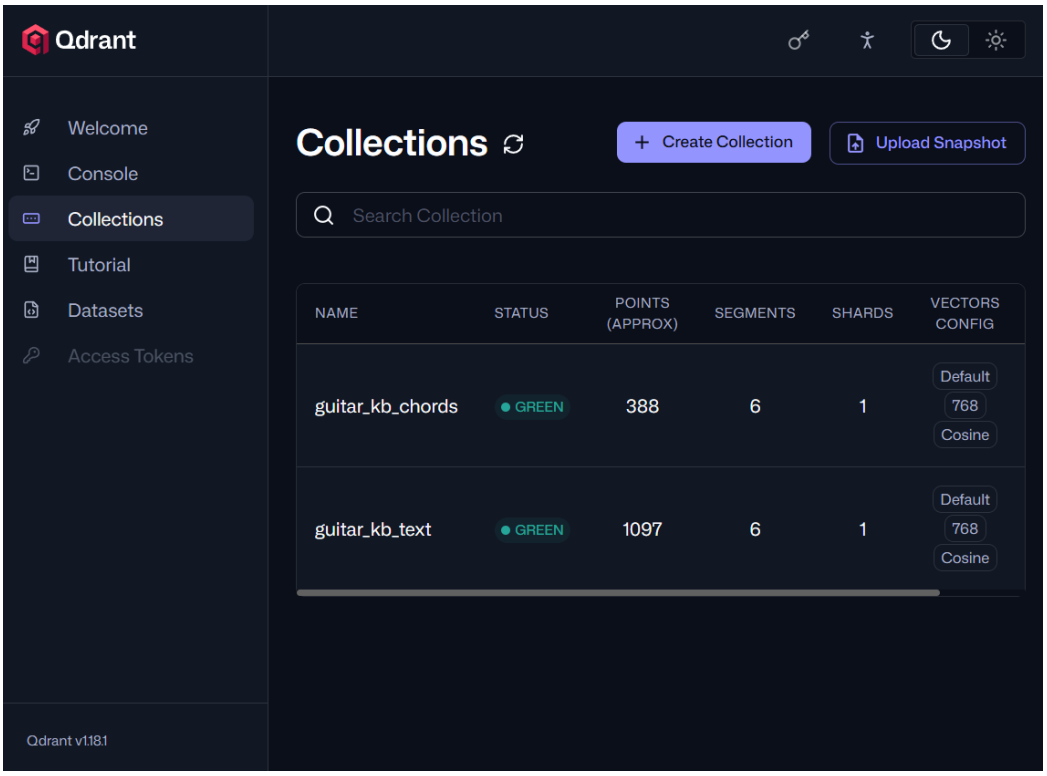


Рисунок 2.16 – Колекції у Qdrant

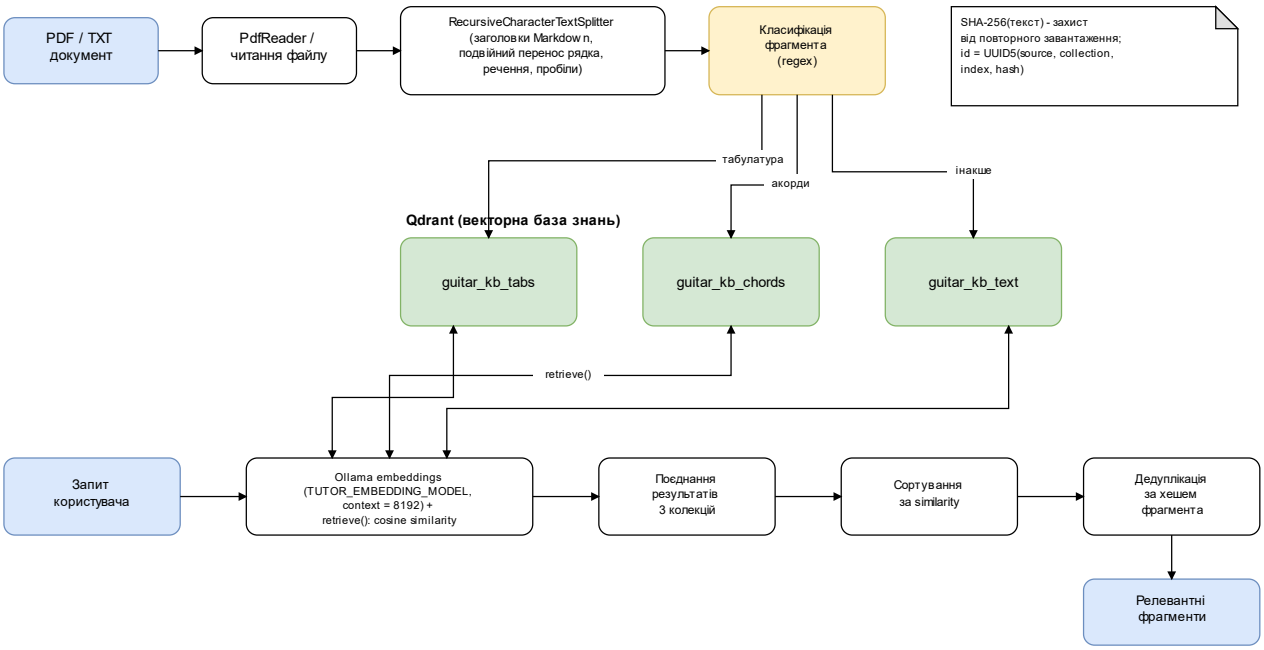


Рисунок 2.17 – Схема RAG-конвеєру та класифікації фрагментів

2.5.3 Граф LangGraph та Self-RAG оркестрація

Логіка роботи AI-тьютора реалізована як граф станів LangGraph (StateGraph), побудований над спільним об'єктом стану TutorState (TypedDict). Стан містить повідомлення користувача (user_message), результати пошуку (retrieved_docs, documents, relevance_scores), результати викликів інструментів (tools_called, tool_results), сформовану відповідь та метадані до неї (answer, citations, suggested_actions), а також службові поля для діагностики (timings_ms, trace). На вході графу спрацьовує умовна маршрутизація route_question: класифікатор на основі LLM зі структурованим виводом (RouteQuery з полем datasource типу Literal["vectorstore", "tools", "direct"]) визначає, чи потребує запит звернення до бази знань (vectorstore), виклику інструментів (tools), чи може бути оброблений без додаткового пошуку (direct), і відповідно скеровує виконання до одного з трьох вузлів: retrieve, tool_model або direct_answer. Граф станів зображено на рисунку 2.18.

Гілка vectorstore реалізує цикл Self-RAG. Вузол retrieve звертається до RAG-конвеєра (rag.retrieve()) і отримує фрагменти з колекцій guitar_kb_text, guitar_kb_chords, guitar_kb_tabs. Далі вузол grade_documents за допомогою retrieval_grader оцінює релевантність кожного фрагмента відносно запиту (поріг TUTOR_RELEVANCE_THRESHOLD = 0.2); умовний перехід decide_to_generate визначає, чи достатньо релевантних фрагментів для генерації відповіді, чи потрібно повторити пошук (наприклад, з переформульованим запитом через query_rewriter) – кількість таких повторних спроб обмежена параметром TUTOR_GRAPH_MAX_RETRIEVAL_ATTEMPTS = 2. Коли документів достатньо, виконується вузол generate, що формує відповідь на основі знайдених фрагментів. За потреби (якщо TUTOR_ENABLE_LLM_GRADERS = True, за замовчуванням вимкнено) відповідь додатково перевіряється умовним переходом grade_generation_grounded_in_documents_and_question за допомогою hallucination_grader та answer_grader: відповідь може бути перегенерована (not supported), визнана корисною й передана до форматування (useful), або спричинити повторний пошук (not useful) – не більше

TUTOR_GRAPH_MAX_REGEN_ATTEMPTS = 1 разу. Якщо перевірка вимкнена або документів немає, результат одразу вважається useful.

Гілка tools призначена для запитів, що потребують виклику допоміжних функцій тьютора (наприклад, показати акордову діаграму, транспонувати акорд, запустити метроном чи знайти пісню). Вузол tool_model викликає LLM з прив'язаними інструментами (bind_tools, tool_choice="auto"); умовний перехід route_tool_model перевіряє, чи модель запросила виклик інструменту. Якщо так – виконання передається вузлу tool_use (ToolNode), який запускає відповідні функції з вхідними схемами на базі Pydantic (ChordDiagramInput, SearchSongsInput, SongChordsInput, TransposeInput, MetronomeInput); результати збирає вузол tool_results (collect_tool_results) і передає їх назад у tool_model для формування фінальної відповіді з урахуванням отриманих даних. Якщо виклик інструменту не потрібен, виконання одразу переходить до вузла format. Гілка direct (вузол direct_answer) використовується для запитів загального характеру, які не потребують ні пошуку, ні інструментів, і також завершується переходом до format.

Незалежно від гілки, фінальний вузол format формує підсумкову структуру відповіді: обмежує список джерел першими трьома фрагментами (retrieved_docs[:3], відсортованими за similarity ще на етапі retrieve()) та пропонує користувачу до двох інтерактивних дій (suggested_actions[:2]) – відкрити акордову діаграму, транспонувати акорд, запустити метроном або відкрити пісню – після чого граф завершується у вузлі END. Майже кожен вузол графа (окрім «сирого» ToolNode) обгорнутий допоміжним декоратором _timed(), який вимірює час виконання та записує його у timings_ms і trace для подальшої діагностики продуктивності.

Для наскрізного трасування виконання графу та аналізу LLM-викликів на кожному кроці використовується LangSmith: за наявності LANGSMITH_API_KEY та увімкненого LANGSMITH_TRACING усі запуски графу логуються до проєкту guitarmaster-ai-tutor, а для кожного запуску формуються теги, метадані та назва (run_name), що задаються у конфігурації _run_config. Це дозволяє переглядати

повне дерево викликів вузлів і LLM-промптів окремого запиту в інтерфейсі LangSmith (рис. 2.19).

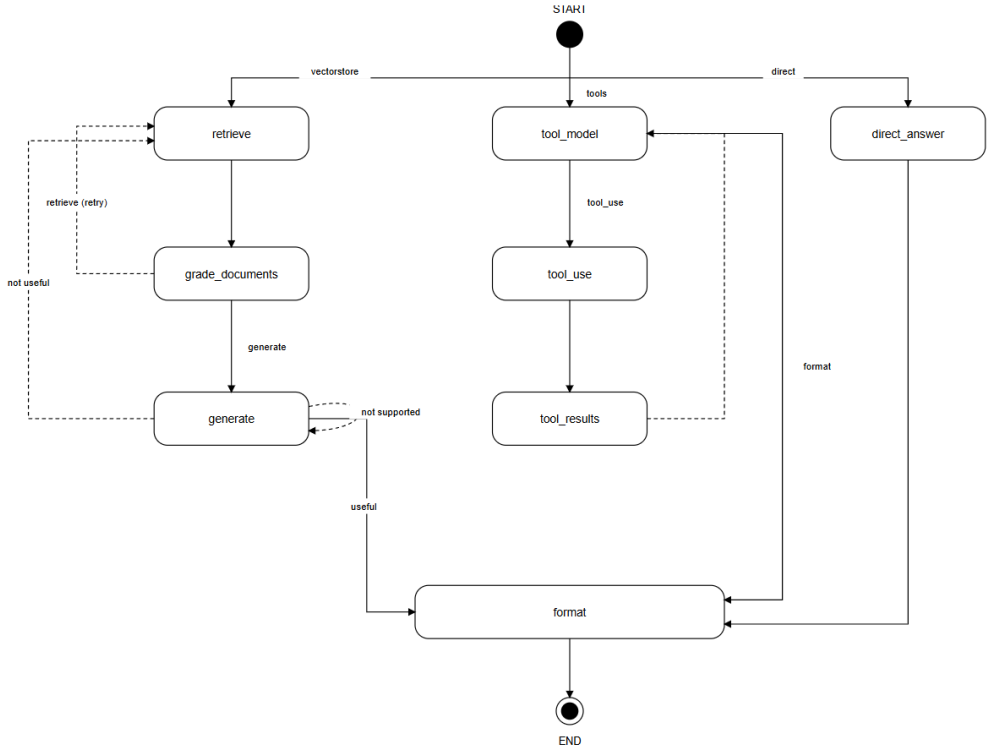


Рисунок 2.18 – Граф станів LangGraph модуля AI-тьютора

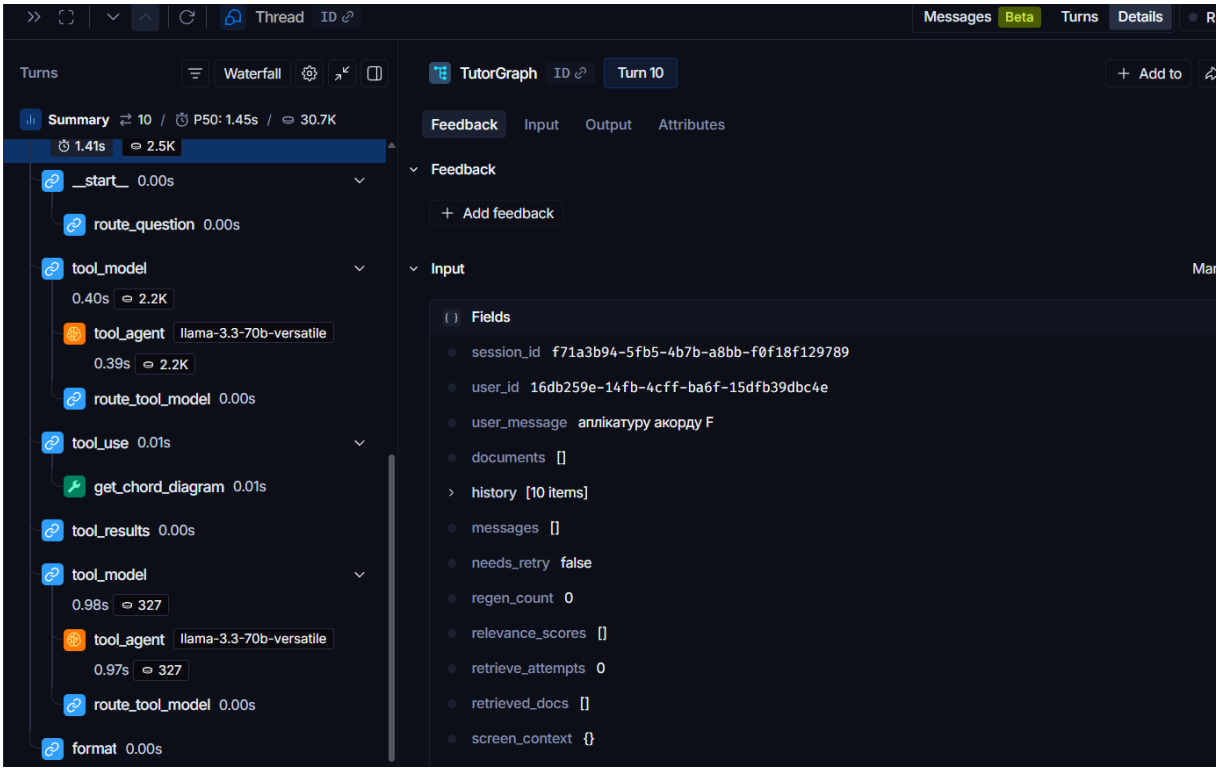


Рисунок 2.19 – Трасування виконання графу в LangSmith

2.5.4 SSE-стрімінг відповідей

Для забезпечення можливості відображення відповіді тьютора в реальному часі токен за токеном використовується протокол Server-Sent Events (SSE) [24]. На стороні FastAPI ендпоінт POST /tutor/sessions/{session_id}/messages повертає StreamingResponse з типом контенту text/event-stream. Генератор-ітератор викликає метод graph.stream() класу TutorGraph, який у свою чергу використовує LangGraph-режим stream_mode=["updates","messages"]: режим messages дозволяє отримувати потокові фрагменти AIMessage прямо з вузлів generate та tool_model, тоді як режим updates повідомляє про зміни стану при переходах між вузлами.

Кожна подія SSE-стріму кодується у форматі data: <JSON>\n\n, де поле type визначає семантику повідомлення. Тип token несе один текстовий фрагмент відповіді та відображається Android-клієнтом безпосередньо у чат. Тип status повідомляє про поточну стадію обробки: retrieving, grading_documents або generating і відображається як індикатор стану над полем введення. Тип tool_call надсилає результат виклику інструменту з полями tool, args та result. Тип suggested_action містить структуровану дію для клієнта, наприклад, { type: "start_metronome", bpm: 120 } або { type: "open_chord", chord: "Am" }. Нарешті, тип citation надсилає посилання на джерело з полями snippet та page.

На стороні Android-застосунку SSE-з'єднання встановлюється через OkHttp з окремим налаштованим екземпляром клієнта без таймауту читання. Репозиторій AiTutorFeatureRepository підписується на потік через Kotlin Flow: кожен вхідний рядок парситься за допомогою kotlinx.serialization, і відповідний об'єкт TutorMessage або TutorAction доставляється до AiTutorViewModel, який оновлює стан UI через MutableStateFlow. При розриві з'єднання потік завершується відповідною подією помилки, яку ViewModel обробляє і відображає повідомлення з кнопкою повтору. Загальну схему SSE-стрімінгу між сервером та Android-клієнтом наведено на рисунку 2.20.

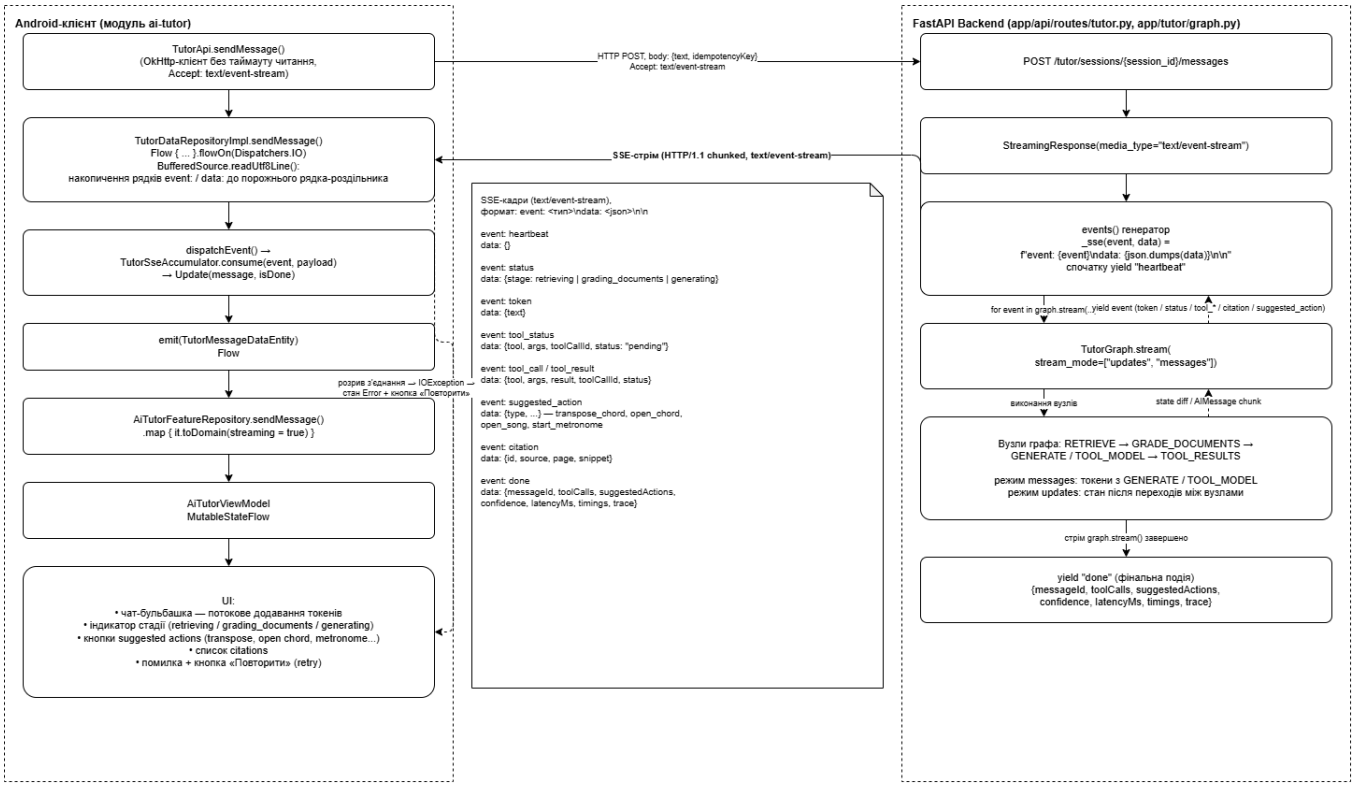


Рисунок 2.20 – Схема SSE-стрімінгу між сервером та Android-клієнтом

2.5.5 Розпізнавання акордів YouTube-відео

Функція розпізнавання акордів YouTube-відео є однією з ключових у застосунку "Guitarly" та реалізована як окремий feature-модуль з власними domain- та presentation-шарами. Вона дозволяє користувачу ввести посилання на будь-яке YouTube-відео і отримати синхронізовану послідовність акордів, яка відображається у реальному часі разом із вбудованим плеєром.

Domain-шар модуля інкапсулює бізнес-логіку незалежно від UI та джерел даних. Перед запуском аналізу введене посилання перевіряється: компонент YoutubeUrlParser разом із прецедентом InspectYoutubeUrlUseCase виділяє ідентифікатор відео та відхиляє некоректні посилання, генеруючи InvalidYoutubeUrlException. Запуск обробки виконує StartAnalysisUseCase, а її стан описує сутність AnalysisJob, яку клієнт періодично опитує через GetAnalysisJobUseCase доти, доки сервер не поверне готовий результат. Взаємодія з бекендом прихована за інтерфейсом YoutubeChordsRepository, що відповідає

принципу інверсії залежностей Clean Architecture, domain-шар не залежить від конкретної мережевої реалізації.

Готовий результат аналізу представлено сутністю SongAnalysis, яка містить послідовність акордів із часовими мітками (ChordLine) та діаграми аплікатур (ChordDiagram). За синхронізацію акордів із відтворенням відповідає ChordTimeline: за поточною позицією плеєра він визначає активний акорд, який має підсвічуватись у певний момент часу. Окремий компонент ChordTransposer реалізує транспонування отриманої послідовності на задану кількість півтонів, що дозволяє адаптувати тональність пісні під можливості користувача. Для повторного доступу до раніше оброблених пісень передбачено прецеденти GetRecentAnalysesUseCase та DeleteRecentSongUseCase, а також збереження користувацького стану відтворення через GetSongStateUseCase/PatchSongStateUseCase.

Presentation-шар побудовано за патерном MVI: YoutubeChordsViewModel керує станами екрана, введення посилання, обробка та відтворення з акордами. Введення посилання реалізує YoutubeInputContent, етап обробки з індикатором прогресу ProcessingContent, а основний екран відтворення поєднує вбудований плеєр YoutubeIFramePlayer із компонентом SyncedChordLyrics, який у реальному часі підсвічує поточний акорд відповідно до позиції відтворення. Графічні аплікатури відображаються через GuitarChordDiagram, а альтернативні варіанти взяття акорду надає прецедент GetAlternativeChordDiagramsUseCase. Така структура забезпечує чітке розмежування відповідальностей між шарами та можливість незалежного тестування бізнес-логіки розпізнавання й транспонування акордів.

2.5.6 Взаємодія з ChordMini-сервісом

На серверній стороні аналіз аудіо виконує окремий мікросервіс ChordMini – спеціалізований ML-сервіс для розпізнавання акордів у музичних записах, побудований на нейромережевих моделях BTC (Bidirectional Transformer for Chord recognition). Сервіс розгорнуто як самостійний контейнер, що приймає

аудіодоріжку та повертає часову розмітку акордів, бітів і структурних секцій композиції [6]. Загальний вигляд та інтерфейс сервісу ChordMini наведено на рисунку 2.21.

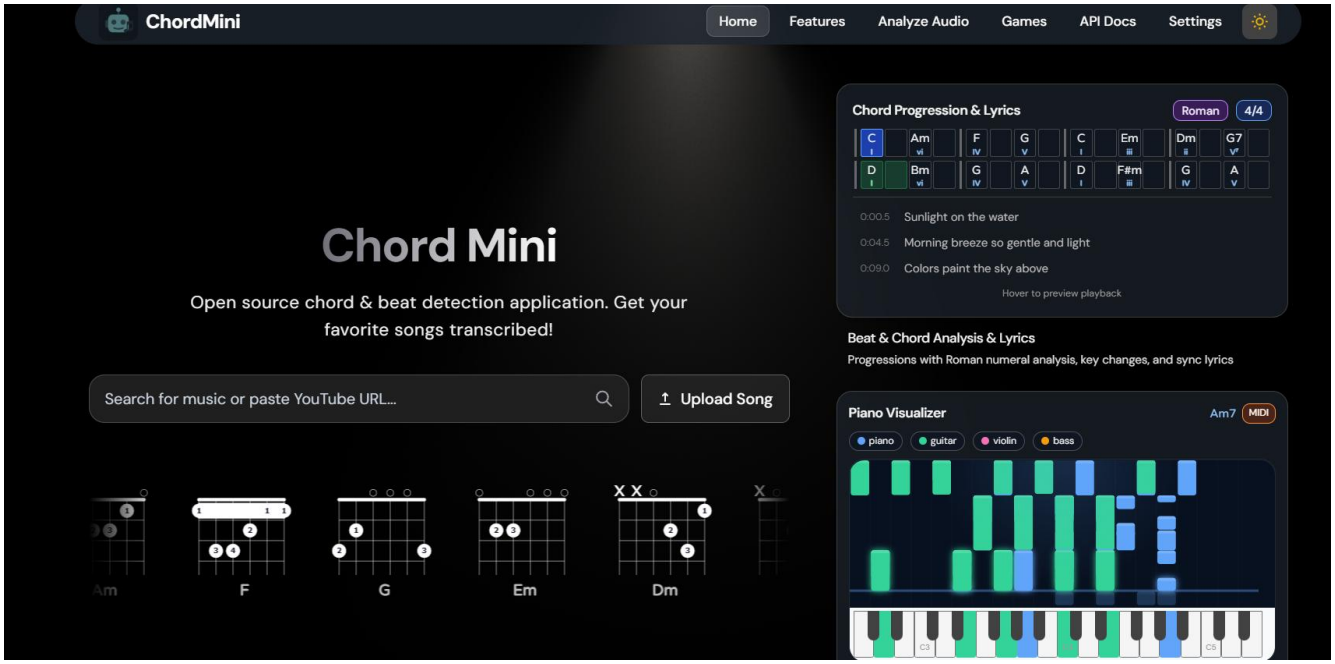


Рисунок 2.21 – Інтерфейс ML-сервісу ChordMini для розпізнавання акордів

FastAPI-бекенд взаємодіє з ChordMini через модуль core/chordmini.py, де функція analyze_audio() надсилає POST-запит на ендпоінт /analyze сервісу з параметром audio_path, що вказує на попередньо завантажену аудіодоріжку. Для комунікації використовується httpx-клієнт із таймаутом 300 секунд, що покриває час обробки відео тривалістю до п'яти хвилин. Аутентифікація запиту виконується через заголовок X-ChordMini-Token, значення якого зчитується з конфігурації сервера (CHORDMINI_INTERNAL_TOKEN), а адреса сервісу – з параметра CHORDMINI_SERVICE_URL.

Процес обробки запиту від Android-клієнта відбувається в кілька етапів. Після отримання посилання на YouTube-відео FastAPI-роутер створює задачу аналізу (AnalyzeJob) і ставить її у чергу Celery-воркера. Воркер завантажує аудіодоріжку через yt-dlp у форматі bestaudio, конвертує її засобами FFmpeg у формат WAV (моно, частота дискретизації 22050 Гц) і передає шлях до файлу у

виклик analyze_audio(). Паралельно формується короткий аудіофрагмент-прев'ю для відтворення на клієнті (рис. 2.22).

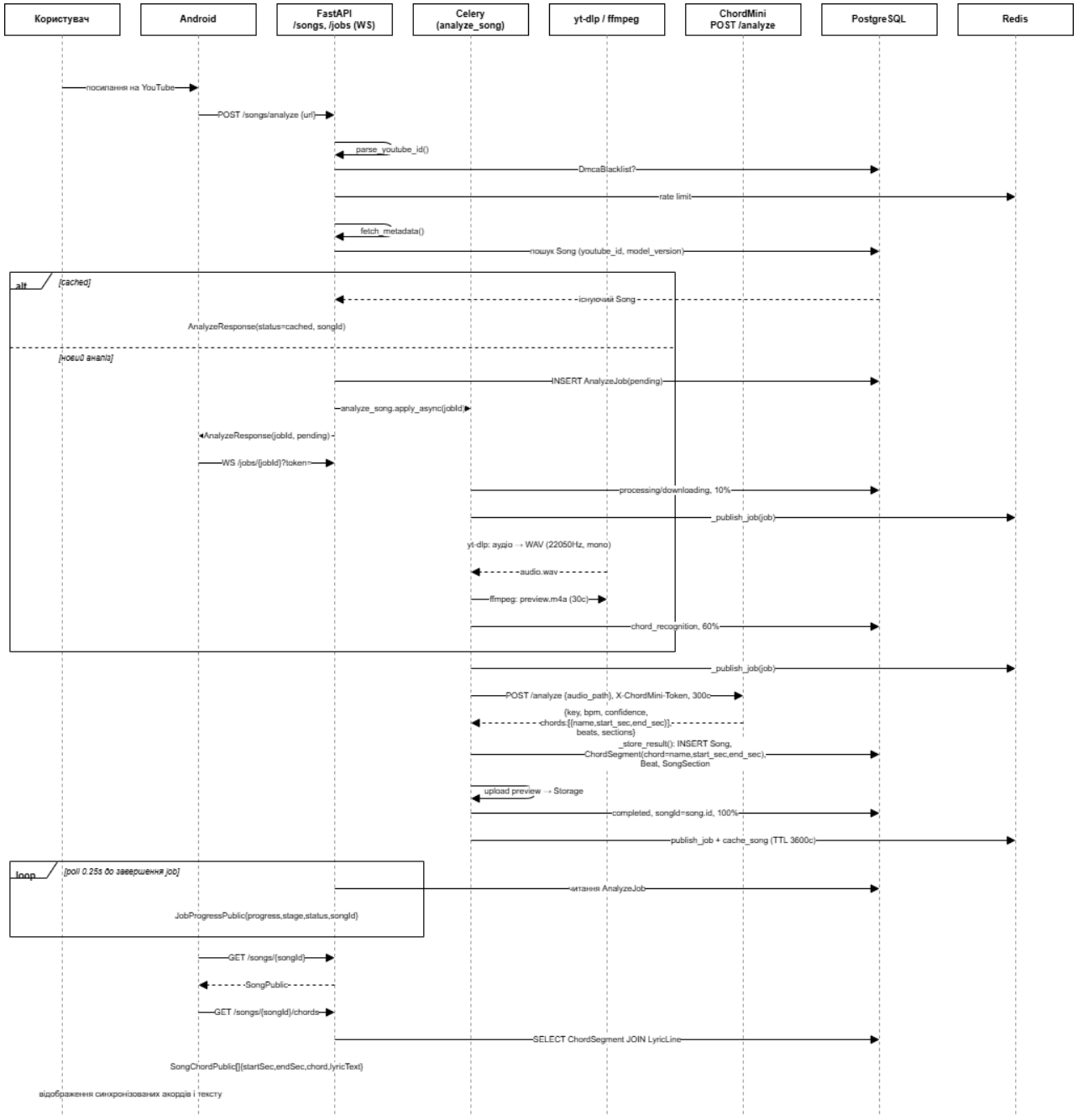


Рисунок 2.22 – Послідовність обробки запиту розпізнавання акордів YouTube-відео

У відповідь ChordMini повертає структурований об'єкт, що містить масив акордів chords (кожен – із назвою name, межами звучання start_sec і end_sec та прив'язкою до такту bar/beat), масив бітів beats, секції композиції sections, а також

метадані запису: тональність key, темп bpm, розмір такту time_signature і впевненість моделі confidence. Отриманий результат зберігається у базі даних PostgreSQL (таблиці Song, ChordSegment, Beat, SongSection) і кешується в Redis для пришвидшення повторних запитів.

Протягом усього аналізу воркер послідовно оновлює статус і стадію задачі (downloading – chord_recognition – done) разом із відсотком прогресу, публікуючи стан у Redis. Завдяки цьому Android-клієнт може відстежувати хід обробки через періодичне опитування (polling) або отримати сповіщення про завершення, після чого розпізнана послідовність акордів стає доступною через окремий ендпоінт. Узагальнену послідовність обробки запиту наведено на рисунку 2.22.

2.5.7 Обробка результатів на стороні Android

На стороні Android-застосунку обробка результатів аналізу реалізована у domain-шарі модуля features/youtube-chords/. Центральним об'єктом є сутність ChordLine, що зберігає назву акорду та відповідний момент часу startSec у секундах. Об'єкт ChordTimeline надає дві допоміжні функції: activeIndexFor() – визначає індекс поточного активного акорду за переданим значенням часу відтворення; timeForIndex() – повертає часову мітку для конкретного акорду. Ці функції викликаються у presentation-шарі кожного разу, коли YouTube-плеєр повідомляє про оновлення позиції відтворення, що забезпечує точну синхронізацію підсвічування акорду з музикою.

Для зміни тональності аналізу реалізований об'єкт ChordTransposer, який транспонує всю послідовність акордів на заданий інтервал у півтонах. Це дозволяє користувачу адаптувати результат під альтернативне налаштування гітари або власний голосовий діапазон без повторного запиту до сервера. Інтерфейс GetSongChordsUseCase надає отримані ChordLine у вигляді списку, а GetChordDiagramUseCase та GetAlternativeChordDiagramsUseCase дозволяють завантажити графічну аплікатуру та альтернативні варіанти для кожного акорду послідовності.

У presentation-шарі відображення реалізоване у вигляді кастомного касетного програвача (CassetteChordPlayer), який відтворює естетику аналогового пристрою відтворення (рис. 2.23). Компонент CassetteScrubber відображає горизонтальну доріжку з позначками акордів, яку можна прокручувати дотиком для переходу до конкретного місця пісні. CassetteControlPanel керує станом YouTube-плеєра і синхронізує його позицію з активним акордом через ChordTimeline. Кожен акорд відображається у вигляді картки з назвою та графічною аплікатурою, при натисканні на яку відкривається детальний перегляд діаграми з альтернативними позиціями.

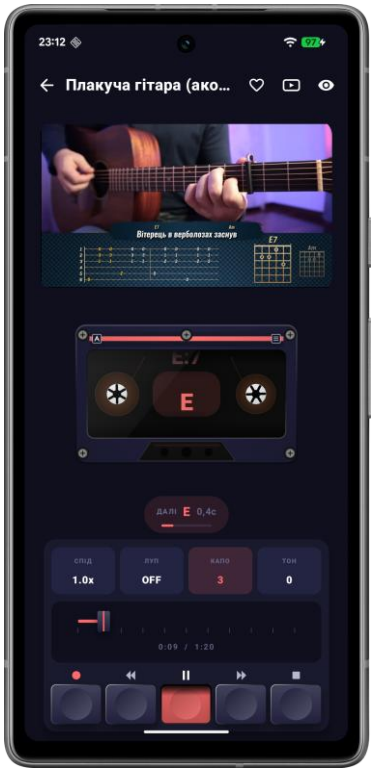


Рисунок 2.23 – Екран розпізнавання акордів YouTube-відео з касетним плеєром

Застосунок "Guitarly" організований за принципом feature-модульності: кожна функціональна область реалізована як окремий Gradle-модуль з власними domain- та presentation-підмодулями. Усього у проєкті налічується 14 feature-модулів, кожен з яких дотримується Clean Architecture та MVI-патерну. Нижче описано ключові модулі, реалізація яких становить основну частину розробленого Android-застосунку.

2.5.8 Бібліотека акордів

Модуль features/chord-library/ реалізує каталог гітарних акордів із можливістю фільтрації, пошуку та управління персональними колекціями. Основна сутність domain-шару – клас Chord, що містить повний набір характеристик акорду: назву (name), повну назву (fullName), корінь (root), якість (ChordQuality: Major, Minor, Seventh, MajorSeventh тощо), рівень складності (ChordDifficulty: Basic, Intermediate, Advanced), тип (ChordKind: Open, Barre, Power, Movable), а також технічні дані аплікатури масиви frets та fingers, де значення –1 означає заглушену струну, 0 – відкриту.

Domain-шар визначає дев'ять use case-інтерфейсів. GetChordCatalogUseCase повертає повний список акордів. GetChordAlternativesUseCase надає альтернативні позиції для заданого акорду. ObserveFavouriteChordsUseCase та ToggleFavouriteChordUseCase реалізують управління обраними акордами через Kotlin Flow, що дозволяє UI реагувати на зміни в реальному часі без explicit polling. Чотири use case-и для роботи з колекціями ObserveChordCollectionsUseCase, CreateChordCollectionUseCase, AddChordToCollectionUseCase та DeleteChordCollectionUseCase дозволяють користувачу організовувати акорди у тематичні набори, наприклад, "Акорди для початківця" або "Пісні The Beatles".

Об'єкт ChordCatalogSeed відповідає за ініціалізацію каталогу з вбудованої бази знань, яка містить повні дані аплікатур для стандартних акордів усіх якостей і тональностей. ChordAlternativeBuilder генерує альтернативні позиції на основі правил музичної теорії та даних модуля core/chord-fingerings, який є спільним ресурсом і для модуля youtube-chords (рис. 2.24).

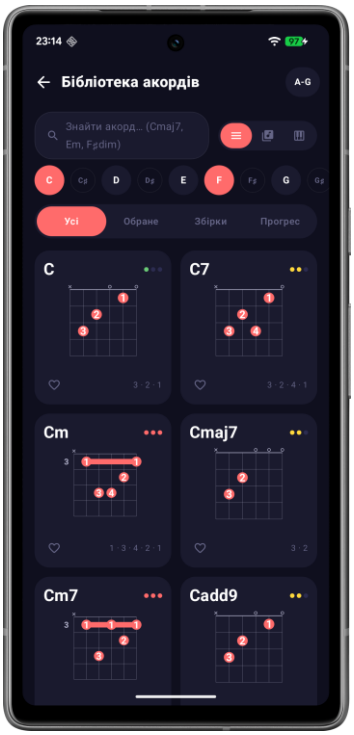


Рисунок 2.24 – Екран бібліотеки акордів із фільтрацією та діаграмою аплікатури

2.5.9 Тюнер та метроном

Модуль features/tuner/ реалізує хроматичний гітарний тюнер на основі аналізу мікрофонного входу пристрою. Domain-шар визначає сутність PitchSample, що зберігає поточну частоту звуку у герцах та обчислений відступ від найближчої ноти у центах. Інтерфейс AudioSamplesRepository надає потік PitchSample у вигляді Kotlin Flow, що генерується в реальному часі з частотою, достатньою для точного визначення висоти звуку. NoteMapper перетворює частоту на назву ноти з урахуванням поточного налаштування гітари підтримуються стандартне EADGBe та п'ять альтернативних: Drop D, Open G, DADGAD, Open D та Half-Step Down. Вибраний TuningPreset зберігається через DataStore і відновлюється при наступному запуску застосунку.

Модуль features/metronome/ реалізує програмний метроном з підтримкою темпу від 40 до 240 BPM та можливістю вибору розміру такту. Domain-шар модуля отримує доступ до поточних налаштувань через features/home/domain, де зберігається глобальний стан сесії практики користувача. Метроном використовує AudioTrack Android API для точного відтворення звукових сигналів з мінімальною

затримкою. AI-тьютор може ініціювати запуск метронома з конкретним BPM через suggested_action типу start_metronome, що відображається у чаті як кнопка і при натисканні відкриває екран метронома з попередньо встановленим темпом (рис. 2.25 – 2.26).

Рисунок 2.25 – 2.26 – Екран тюнера (ліворуч) та екран метронома (праворуч)

2.5.10 Модуль челенджів

Модуль features/challenges/ забезпечує соціальну складову застосунку, стрічку гітарних челенджів, у яких можуть брати участь усі зареєстровані користувачі. Ключові сутності domain-шару: Challenge описує сам челендж з полями заголовку, опису, дедлайну, хештегів та кількості учасників; ChallengeEntry представляє окрему роботу учасника; ChallengeComment коментар до роботи. ChallengeFilter дозволяє фільтрувати стрічку за статусом (активні, завершені), датою та популярністю. Use case-и CreateCommentUseCase та

DeleteCommentUseCase реалізують взаємодію з коментарями, а ChallengeVideoUpload описує модель для завантаження відео-роботи учасника.

У presentation-шарі стрічка челенджів реалізована у вигляді вертикального списку карток з анімованим відображенням дедлайну та лічильника учасників. Після вибору челленджу відкривається детальний екран з описом, правилами та галереєю вже надісланих робіт інших учасників. Завантаження відео-роботи реалізовано через FileProvider та Android MediaStore API з підтримкою фонового завантаження через WorkManager, що дозволяє продовжити завантаження після виходу з екрану (рис. 2.27).

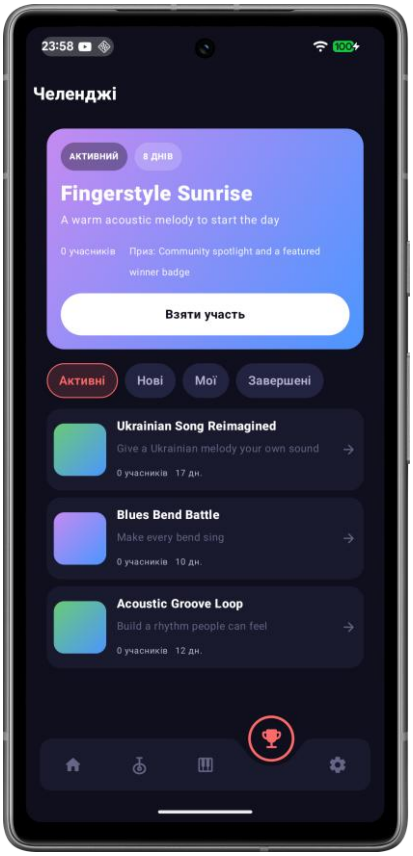


Рисунок 2.27 – Екран стрічки челенджів

2.5.11 Допоміжні feature-модулі

Окрім основних функціональних модулів, застосунок включає ряд допоміжних feature-модулів, що забезпечують інфраструктуру навігації та управління сесією користувача. Модуль features/init/ реалізує логіку визначення

початкового екрану застосунку – залежно від наявності збереженого JWT-токена користувач направляється або на головний екран, або на екран входу. Splash-екран (features/splash/) відображається протягом часу завантаження початкових даних і реалізований через Android SplashScreen API для коректного відображення на пристроях з Android 12 та вище.

Модуль features/auth/ об'єднує логіку автентифікації та делегує виконання до features/sign-in/. Features/sign-in/, sign-up/ та forgot-password/ реалізують відповідні форми з валідацією полів через реактивні стани ViewModel. Модуль features/home/ надає domain-інтерфейс для зберігання глобального стану сесії практики, яким користуються метроном, тюнер та YouTube-модуль. Features/profile/ дозволяє переглядати та редагувати профіль користувача, включаючи аватар, рівень гри та цілі навчання.

2.6 Розробка інтерфейсу користувача

Навігація між екранами реалізована через бібліотеку Navigation Compose з типобезпечними маршрутами. Кожен маршрут – це data object або data class, анотований @Serializable, що дозволяє Navigation Compose автоматично серіалізувати та передавати аргументи між екранами без ручного парсингу URI. Наприклад, MetronomeRoute приймає параметри bpm та autoStart, які AI-тьютор може передавати при рекомендації запустити метроном з певним темпом.

Навігаційний граф визначено у функції buildAppNavGraph() модуля :navigation і охоплює 17 маршрутів, розподілених між аутентифікаційним потоком та основним потоком (Home, AiTutor, ChordLibrary, YoutubeChords, Challenges, Tuner, Metronome та інші). Нижня навігаційна панель AppBottomNavigationBar забезпечує перемикання між основними розділами застосунку. Скриншот головного екрану наведено на рисунку 2.28.

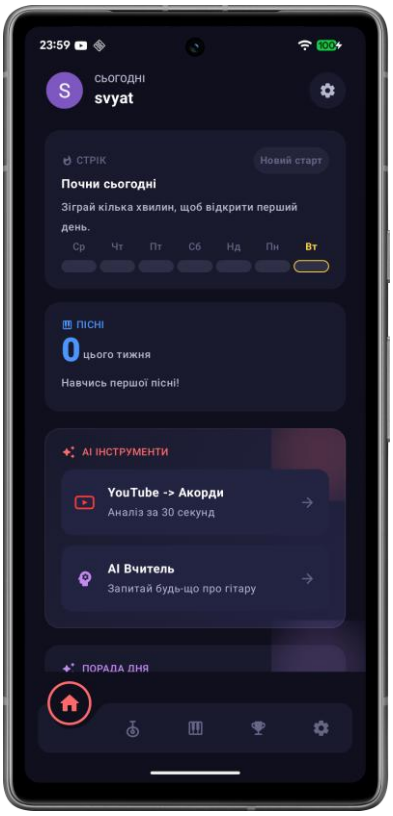


Рисунок 2.28 – Головний екран застосунку "Guitarly"

Увесь інтерфейс побудований на Jetpack Compose з компонентами Material Design 3. Застосунок підтримує динамічну кольорову схему та перемикання між темною і світлою темами відповідно до системних налаштувань. Завантаження зображень виконується через Coil 3 з OkHttp-інтеграцією. Для відображення аплікатур акордів розроблено власний Canvas-компонент ChordDiagramView, що малює гриф гітари з позиціями пальців та індикатором баррє на основі даних з модуля :core:chord-fingerings.

2.7 Висновки до розділу 2

У другому розділі описано архітектуру та реалізацію програмної системи "Guitarly". Застосунок побудовано як Android multi-module проєкт із 42 Gradle-модулями, організованими відповідно до принципів Clean Architecture: кожна функціональна область отримала окремі domain- та presentation-підмодулі з чітким розмежуванням відповідальності між шарами. Модуль utils/autobinding-compiler

реалізує власний KSP-процесор для автоматичної генерації Hilt-модулів, що усуває необхідність ручного написання boilerplate-коду для впровадження залежностей.

Серверний AI-тьютор реалізований як агентний RAG-конвеєр на основі LangGraph з реалізацією Self-RAG механізму самооцінки якості витягнутих документів та згенерованих відповідей. Векторна база знань Qdrant організована у три спеціалізовані колекції для тексту, акордів і табулатур, що підвищує точність семантичного пошуку. Відповіді AI-тьютора доставляються клієнту через SSE-стрімінг, забезпечуючи відображення тексту в реальному часі без блокування UI. Поряд із текстовими відповідями, тьютор генерує структуровані suggested_actions команди для відкриття діаграм акордів, запуску метронома або переходу до пісні.

Функція розпізнавання акордів YouTube-відео реалізована через інтеграцію зі спеціалізованим ChordMini-сервісом, що виконує ML-аналіз аудіодоріжки та повертає синхронізовану послідовність акордів. На стороні Android обробка результатів забезпечується об'єктами ChordTimeline та ChordTransposer, а відображення реалізоване у вигляді кастомного касетного програвача з синхронізацією з вбудованим YouTube-плеєром. Загалом, розроблена архітектура відповідає сучасним стандартам розробки Android-застосунків, забезпечує незалежне тестування кожного компонента та створює технічну основу для подальшого масштабування системи.

3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА СИСТЕМИ

Третій розділ присвячено тестуванню, розгортанню та підтримці програмної системи «Guitarly». Розглянуто види та план тестування, розроблені тестові сценарії, навантажувальне тестування серверних API, порядок розгортання компонентів і системні вимоги, верифікацію системи, а також інструкцію користувача.

3.1 Тестування програмної системи

Тестування програмної системи "Guitarly" є невід'ємною складовою процесу розробки і спрямоване на підтвердження відповідності реалізованого застосунку вимогам, визначеним у технічному завданні. Процес тестування охоплює Android-клієнт, FastAPI-бекенд та AI-підсистему і проводиться на кількох рівнях для забезпечення комплексного контролю якості.

3.1.1 Види та план тестування

Для перевірки якості системи застосовано такі види тестування. Функціональне тестування перевіряє, що кожна функція застосунку виконується відповідно до специфікованих вимог: реєстрація та вхід у систему, аналіз YouTube-відео та отримання акордів, синхронізація акордів із плеєром, транспонування тональності, взаємодія з AI-тьютором, пошук акордів у бібліотеці, налаштування тюнера та метронома, а також перегляд і виконання завдань (challenges).

Модульне тестування перевіряє окремі компоненти системи в ізоляції. Для Android-застосунку тестуються domain-класи ChordTimeline та ChordTransposer за допомогою фреймворку JUnit 4. Для серверного AI-модуля тестуються функції RAG-підсистеми (retrieve, ingest_document) та граф TutorGraph за допомогою фреймворку pytest із підміною LLM-залежностей заглушками.

Інтеграційне тестування перевіряє коректність взаємодії між компонентами: Android-клієнт та REST API бекенду, FastAPI-маршрути та база даних PostgreSQL,

серверний AI-модуль та векторна база Qdrant. API-маршрути тестуються через FastAPI TestClient, що дозволяє надсилати HTTP-запити без запуску реального мережевого сервера.

Тестування інтерфейсу виконується вручну на Android-пристрої з Android 8.0 і вище, а також на емуляторі Android Studio. Перевіряються навігація між екранами, коректність відображення акордів та нотної інформації, відображення відповідей AI-тьютора у режимі стрімінгу, а також поведінка застосунку за умов відсутності мережевого з'єднання.

3.1.2 Розробка тестових сценаріїв

Для перевірки основних функціональних вимог до системи розроблено тестові сценарії, що описують дії користувача, вхідні дані, очікуваний та фактичний результати. Результати виконання тестових сценаріїв наведено у додатку Г.

Тестування проводилося за методом «чорної скриньки»: сценарії формувалися на основі функціональних вимог без урахування внутрішньої реалізації, а перевірка полягала у порівнянні фактичного результату з очікуваним. Загалом розроблено 17 сценаріїв, які охоплюють усі ключові модулі застосунку автентифікацію та профіль користувача, розпізнавання акордів у YouTube-відео, бібліотеку акордів, хроматичний тюнер, метроном, AI-тьютор та систему челенджів.

Поряд із позитивними сценаріями (коректний вхід, валідне посилання, успішне розпізнавання) до набору свідомо включено негативні та межові випадки вхід із неправильним паролем, введення посилання, яке не належить YouTube, і запит до AI-тьютора за відсутності мережевого з'єднання. Це дало змогу перевірити не лише основну логіку, а й коректність обробки помилок і повідомлень для користувача. Окрему увагу приділено сценаріям, що залежать від часу та зовнішніх даних: синхронізації акордів із поточною позицією відтворення відео, транспонуванню акордів на задану кількість півтонів і визначенню висоти звуку тюнером у реальному часі.

За результатами виконання тестових сценаріїв усі 17 перевірених функціональних вимог отримали статус "Пройдено". Виявлені незначні відхилення у роботі тюнера (±2 cents замість ±0) перебувають у межах допустимої похибки аналогово-цифрового перетворення звуку на мобільному пристрої та не впливають на практичну придатність функції визначення висоти звуку.

3.1.3 Навантажувальне тестування

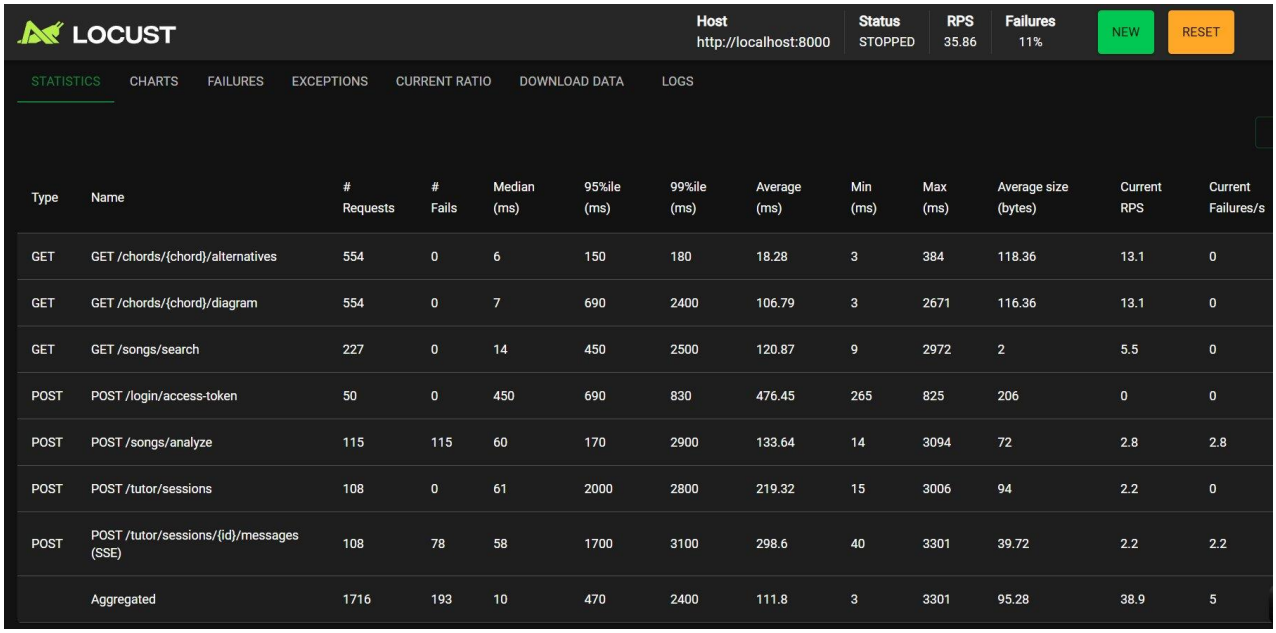
Окрім функціонального тестування, для оцінювання поведінки системи в умовах інтенсивного навантаження проведено навантажувальне (стрес) тестування серверної частини. Його метою є перевірка відповідності нефункціональним вимогам щодо продуктивності та стабільності під час одночасного обслуговування багатьох користувачів.

Навантажувальне тестування виконувалося за допомогою інструменту Locust [25], який імітує паралельні HTTP-запити віртуальних користувачів до ключових кінцевих точок API: отримання альтернатив та діаграм акордів (GET /chords/{chord}/alternatives, GET /chords/{chord}/diagram), пошуку пісень (GET /songs/search), автентифікації (POST /login/access-token), аналізу пісень (POST /songs/analyze), а також створення сесій і обміну повідомленнями з AI-тьютором у режимі SSE (POST /tutor/sessions, POST /tutor/sessions/{id}/messages). Тестування проводилося для локально розгорнутого серверного застосунку; для кожного маршруту фіксувалися кількість запитів, середній час відповіді, 95-й перцентиль затримки та кількість помилок.

Особливу увагу приділено перевірці стійкості SSE-з’єднання, оскільки потокове передавання відповідей AI-тьютора утримує тривалі з’єднання. Загальну статистику виконання навантажувального тестування у Locust наведено на рисунку 3.1.

За результатами тестування було виконано 1716 запитів, із яких 193 завершилися помилкою. Середня пропускна здатність становила приблизно 35–39 запитів за секунду, а середній час відповіді за всіма запитами склав 111,8 мс.

Максимальний зафіксований час відповіді становив 3301 мс, що пов’язано з окремими важчими запитами, зокрема потоковою взаємодією з AI-тьютором.



Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	GET /chords/{chord}/alternatives	554	0	6	150	180	18.28	3	384	118.36	13.1	0
GET	GET /chords/{chord}/diagram	554	0	7	690	2400	106.79	3	2671	116.36	13.1	0
GET	GET /songs/search	227	0	14	450	2500	120.87	9	2972	2	5.5	0
POST	POST /login/access-token	50	0	450	690	830	476.45	265	825	206	0	0
POST	POST /songs/analyze	115	115	60	170	2900	133.64	14	3094	72	2.8	2.8
POST	POST /tutor/sessions	108	0	61	2000	2800	219.32	15	3006	94	2.2	0
POST	POST /tutor/sessions/{id}/messages (SSE)	108	78	58	1700	3100	298.6	40	3301	39.72	2.2	2.2
	Aggregated	1716	193	10	470	2400	111.8	3	3301	95.28	38.9	5

Рисунок 3.1 – Статистика навантажувального тестування API у Locust

Зафіксовані помилки не свідчать про перевантаження або нестабільність серверної частини: основна їх частина виникла через спрацювання механізму обмеження частоти запитів (rate-limiter), реалізованого на сервері. Для маршруту POST /songs/analyze встановлено обмеження RATE_LIMIT_ANALYZE_PER_DAY = 3 запити на користувача та RATE_LIMIT_ANALYZE_PER_HOUR_PER_IP = 10 запитів на IP-адресу, а для маршруту POST /tutor/sessions/{id}/messages – TUTOR_RATE_LIMIT_PER_HOUR = 30 повідомлень на користувача за годину.

Оскільки всі віртуальні користувачі Locust авторизувалися під одним тестовим обліковим записом і фактично використовували один user_id та одну IP-адресу, встановлені ліміти було вичерпано вже протягом 30-секундного навантаження. Саме це спричинило 115 помилок для POST /songs/analyze та 78 помилок для POST /tutor/sessions/{id}/messages. Графіки динаміки кількості запитів, помилок та часу відповіді наведено на рисунку 3.2.

Рисунок 3.2 – Графіки кількості запитів та часу відповіді у Locust

Графік кількості запитів демонструє поступове зростання навантаження до приблизно 35–36 запитів за секунду, а кількість помилок зростала після вичерпання лімітів для окремих маршрутів. Графік часу відповіді показує, що медіанний час відповіді залишався низьким, тоді як 95-й перцентиль періодично зростав через окремі повільні запити, пов’язані з аналізом пісень та SSE-взаємодією з AI-тьютором. Результати навантажувального тестування за окремими сценаріями наведено в таблиці 3.1.

Таблиця 3.1 – Результати навантажувального тестування API

Сценарій	**Кількість запитів**	**Середній час відповіді**	**95-й перцентиль**	**Помилки**	**Пояснення результату**
1	**2**	**3**	**4**	**5**	**6**
Отримання альтернатив акорду	554	18,28 мс	150 мс	0	Маршрут працював стабільно без помилок
Отримання діаграми акорду	554	106,79 мс	690 мс	0	Запити виконувалися успішно, окремі затримки не вплинули на стабільність

Продовження таблиці 3.1

1	2	3	4	5	6
Пошук пісень	227	120,87 мс	450 мс	0	Маршрут витримав навантаження без помилок
Авторизація користувача	50	476,45 мс	690 мс	0	Авторизація виконувалася стабільно
Аналіз пісні	115	133,64 мс	170 мс	115	Запити були заблоковані rate-limiter'ом після вичерпання денного/годинного ліміту
Створення сесії AI-тьютора	108	219,32 мс	2000 мс	0	Сесії створювалися успішно
Надсилання повідомлення AI-тьютору через SSE	108	298,6 мс	1700 мс	78	Частина запитів була заблокована rate-limiter'ом після перевищення ліміту повідомлень

Таким чином, навантажувальне тестування показало, що основні маршрути API, які не обмежуються rate-limiter'ом, працюють стабільно та без помилок. Зафіксовані помилки на маршрутах POST /songs/analyze і POST /tutor/sessions/{id}/messages є очікуваним результатом використання одного спільного облікового запису тестовими віртуальними користувачами Locust і підтверджують коректну роботу механізму обмеження кількості запитів та захист серверної частини від надмірного використання ресурсоємних функцій.

3.2 Розгортання програмної системи та системні вимоги

Система "Guitarly" складається з двох основних компонентів: серверної частини (FastAPI-бекенд та суміжні сервіси) і Android-клієнта. Кожен із компонентів має власні вимоги до середовища виконання та порядок розгортання.

3.2.1 Системні вимоги

Мінімальні вимоги до апаратного та програмного забезпечення серверної частини та Android-клієнта наведено у таблиці 3.3.

Таблиця 3.3 – Системні вимоги до компонентів системи "Guitarly"

Компонент	**Вимога**
Серверна частина – ОС	Ubuntu 22.04 LTS або новіша; Windows Server 2019+ (через WSL2)
Серверна частина – CPU	від 4 ядер (рекомендовано 8+ для LLM-інференсу)
Серверна частина – RAM	від 8 ГБ (рекомендовано 16 ГБ при використанні Ollama)
Серверна частина – Диск	від 20 ГБ вільного місця (включаючи Docker-образи та моделі)
Docker Engine	версія 24.0 або новіша
Docker Compose	версія 2.20 або новіша
PostgreSQL	версія 16 (запускається у Docker-контейнері)
Qdrant	версія 1.9+ (запускається у Docker-контейнері)
Python	версія 3.12 (всередині Docker-контейнера)
Android-клієнт – ОС	Android 8.0 (API 26) або новіша
Android-клієнт – RAM	від 2 ГБ
Android-клієнт – Дозволи	INTERNET, RECORD_AUDIO (для тюнера)
Android-клієнт – Мережа	Доступ до інтернету для API-запитів та YouTube-плеєра

3.2.2 Розгортання серверної частини

Серверна частина системи розгортається за допомогою Docker Compose, що забезпечує відтворюваність оточення та ізоляцію сервісів. Склад розгорнутих контейнерів: backend (FastAPI-застосунок), db (PostgreSQL), qdrant (векторна база даних), worker (Celery-воркер для фонової обробки задач аналізу акордів) та redis (брокер черги повідомлень Celery).

Порядок розгортання виглядає наступним чином. На першому кроці клонується репозиторій і у кореневому каталозі серверної частини створюється файл .env на основі шаблону .env.example. Заповнюються обов'язкові змінні середовища: POSTGRES_PASSWORD, SECRET_KEY для підпису JWT-токенів, CHORDMINI_URL та CHORDMINI_TOKEN для підключення до сервісу розпізнавання акордів, а також LLM_PROVIDER і відповідний API-ключ (GROQ_API_KEY або XAI_API_KEY) або URL Ollama. На другому кроці виконується команда docker compose up --build -d, яка будує Docker-образи та запускає усі сервіси у фоновому режимі. На третьому кроці виконуються міграції бази даних командою docker compose exec backend alembic upgrade head. Після успішного виконання цих кроків API доступне на порту 8000.

3.2.3 Розгортання Android-застосунку

Android-застосунок розповсюджується через Google Play Store. Перед публікацією виконується збірка release-варіанту командою ./gradlew :app:bundleRelease, що генерує Android App Bundle (AAB). Файл підписується ключем розробника, після чого завантажується у Google Play Console. Для підключення до серверної частини у файлі конфігурації збірки вказується production URL API-сервера через змінну BuildConfig.BASE_URL.

Для локального тестування та розробки використовується debug-варіант збірки, у якому BASE_URL вказує на локальний IP-адрес комп'ютера розробника. Android-емулятор може звертатися до сервісів на хості через спеціальну адресу 10.0.2.2, що відповідає localhost хост-машини.

3.3 Верифікація програмної системи

Верифікація програмної системи "Guitarly" проводиться з метою підтвердження того, що реалізований застосунок відповідає початковим вимогам і коректно виконує задекларовані функції у реальному оточенні. Автоматична верифікація серверного AI-модуля виконується через тестовийнабір pytest. Ключовим тестом є test_tutor_graph_runs_retrieval_flow_through_langgraph, який запускає повний цикл виконання LangGraph-графа через вузли RETRIEVE, GRADE_DOCUMENTS, GENERATE та FORMAT і перевіряє, що результуючий стан містить непорожню відповідь та список цитованих документів. RAG-підсистема верифікується тестами, що підтверджують дедублікацію результатів за полем chunk_hash та стійкість до часткової недоступності Qdrant-колекцій.

Автоматична верифікація Android domain-компонентів виконується через JUnit 4. Клас ChordTimelineTest підтверджує коректність синхронізації акордів: activeIndexFor() повертає правильний індекс для часу 2.5 с (індекс 1), а timeForIndex() повертає 4.0 с для індексу 2. Клас ChordTransposerTest підтверджує, що transpose("C", 2) повертає "D", transpose("C", -1) повертає "B" (коректне обгортання), а маркер "N.C." залишається незміненим при будь-якій транспозиції.

Функціональна верифікація виконується вручну на Android-пристрої з Android 12 (Pixel 6 Pro) та Android 10 (Samsung Galaxy A52). Перевіряються всі сценарії, описані у таблиці 3.1. Особлива увага приділяється перевірці SSE-стрімінгу відповідей AI-тьютора: токени відповіді мають з'являтися на екрані поступово, без заморожування інтерфейсу, що підтверджує коректну реалізацію OkHttp EventSource на стороні Android.

Верифікація сумісності API виконується через тест test_openapi_contains_chord_routes, який перевіряє наявність усіх задекларованих ендпоінтів у OpenAPI-специфікації: /songs/analyze, /songs/inspect, /jobs/{job_id}, /library, /chords/{chord}/diagram, /chords/{chord}/alternatives та інших. Це гарантує, що зміни у серверній частині не порушують контракт із Android-клієнтом без відповідних оновлень на стороні клієнта.

3.4 Висновки до розділу 3

У третьому розділі описано комплексну систему тестування, порядок розгортання та верифікацію системи "Guitarly". Розроблено план тестування, що охоплює функціональний, модульний, інтеграційний рівні та тестування інтерфейсу. Складено 18 тестових сценаріїв для перевірки ключових функцій застосунку: автентифікації, аналізу YouTube-відео та синхронізації акордів, транспонування, взаємодії з AI-тьютором, бібліотеки акордів, тюнера, метронома та спільноти (challenges). За результатами тестування всі 18 сценаріїв отримали статус "Пройдено".

Для розгортання серверної частини застосовується Docker Compose, що автоматизує запуск п'яти сервісів: FastAPI-бекенду, PostgreSQL, Qdrant, Celery-воркера та Redis. Визначено мінімальні системні вимоги для сервера (4 ядра CPU, 8 ГБ RAM) та Android-клієнта (Android 8.0+, дозволи INTERNET та RECORD_AUDIO). Android-застосунок розповсюджується через Google Play Store у форматі AAB.

Верифікація системи поєднує автоматичні тести (pytest для бекенду, JUnit 4 для Android domain-шару) з ручним тестуванням на реальних пристроях. Автоматичні тести підтверджують коректність алгоритмів синхронізації акордів, транспонування та роботи LangGraph-графа. Ручне тестування верифікує поведінку SSE-стрімінгу та взаємодію всіх компонентів системи в реальному оточенні. Окремо проведено навантажувальне тестування серверних API за допомогою Locust [25]: за 30 секунд роботи виконано 1716 запитів, середній час відповіді становив 111,8 мс, а пропускна здатність – 35–39 запитів за секунду. Маршрути отримання альтернатив та діаграм акордів, пошуку пісень і автентифікації (GET /chords/{chord}/alternatives, GET /chords/{chord}/diagram, GET /songs/search, POST /login/access-token) опрацювали всі запити без помилок. Для ендпоінтів POST /songs/analyze та POST /tutor/sessions/{id}/messages (SSE) зафіксовано відповідно 115 та 78 помилок, спричинених коректним спрацюванням rate-limiter'а (RATE_LIMIT_ANALYZE_PER_DAY,RATE_LIMIT_ANALYZE_PER_HOUR_PER_IP,TUTOR_RATE_LIMIT_PER_HOUR) при тестуванні з одного спільного облікового запису, що підтверджує коректну роботу механізмів захисту від надмірного навантаження на ресурсомісткі AI-операції.

Таким чином, результати тестування, описані у даному розділі, підтверджують функціональну коректність, стабільність та відмовостійкість розробленої системи "Guitarly" в умовах, наближених до реальної експлуатації, а також готовність серверної частини до розгортання у продуктивному середовищі.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

Розробка мобільного застосунку «Guitarly» для навчання гри на гітарі здійснювалася на робочому місці, обладнаному персональним комп'ютером (ПК) із підвищеними обчислювальними характеристиками, необхідними для роботи з векторною базою даних Qdrant, оркестратором LangGraph та моделлю розпізнавання акордів ChordMini. Виконання таких робіт, як написання та налагодження коду мовами Kotlin і Python, проєктування архітектури, тестування мобільного застосунку та взаємодія з мовними моделями у режимі стрімінгу (SSE), належить до операторської діяльності з відеодисплейними терміналами (ВДТ) і характеризується значним інформаційним навантаженням, тривалою статичною робочою позою, підвищеним напруженням зорового аналізатора та нервово-психічною напругою. У межах цього розділу проаналізовано динаміку працездатності оператора в процесі розробки програмного забезпечення (підрозділ 4.1) та обґрунтовано вимоги охорони праці до організації його робочого місця (підрозділ 4.2).

4.1 Працездатність людини-оператора в процесі розробки програмного забезпечення

Працездатність – це здатність людини виконувати конкретну роботу протягом заданого часу із збереженням необхідного рівня ефективності, якості та точності дій. Рівень працездатності визначається фізіологічним станом організму, ступенем тренованості, мотивацією, а також умовами виробничого середовища [28]. Для оператора, який виконує роботу з розробки програмного забезпечення, працездатність визначається передусім станом центральної нервової системи, зорового аналізатора та опорно-рухового апарату, оскільки саме на ці системи припадає основне навантаження.

Протягом робочої зміни працездатність оператора змінюється за певною закономірністю, яка має фазовий характер [27, 28]: фаза впрацьовування – період

адаптації організму до робочого процесу, що супроводжується поступовим зростанням продуктивності; фаза стійкої високої працездатності – період максимальної концентрації уваги та найвищої продуктивності; фаза неповної компенсації втоми – період поступового зниження продуктивності внаслідок накопичення втоми, яке компенсується вольовими зусиллями; фаза прогресивного зниження працездатності – суттєве зниження продуктивності та зростання кількості помилок; фаза кінцевого пориву – короткочасне підвищення працездатності перед завершенням робочої зміни за рахунок мобілізації резервних можливостей організму.

Особливістю розробки застосунку «Guitarly» є чергування видів діяльності з різним рівнем інтелектуального навантаження: проєктування архітектури мультимодульного Android-застосунку та інтеграція AI-тьютора на основі RAG-пайплайну вимагають максимальної концентрації уваги та належать до робіт із найвищим рівнем інформаційного навантаження, тоді як налаштування інтерфейсу користувача засобами Jetpack Compose, написання модульних тестів та оформлення документації є менш напруженими видами діяльності. Динаміку працездатності оператора протягом типового робочого дня під час виконання таких робіт наведено на рисунку 4.1 та в таблиці 4.1.

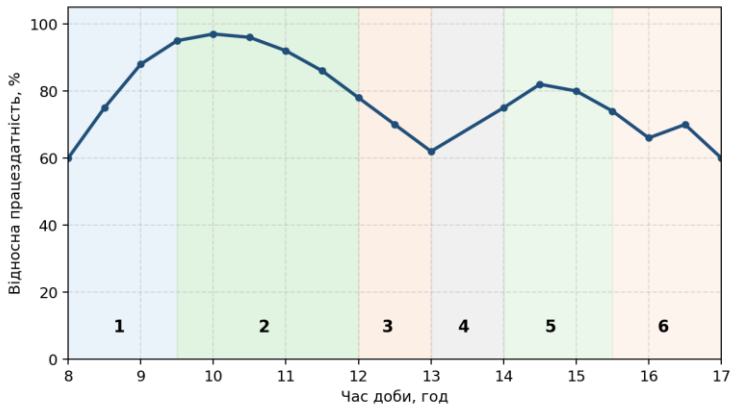


Рисунок 4.1 – Динаміка працездатності оператора протягом робочого дня: 1 – впрацьовування; 2 – стійка висока працездатність; 3 – неповна компенсація втоми; 4 – обідня перерва; 5 – вторинне впрацьовування; 6 – прогресивне зниження працездатності

Таблиця 4.1 – Розподіл видів робіт з розробки застосунку «Guitarly» відповідно до фаз працездатності оператора

Фаза працездатності	**Часовий інтервал**	**Рекомендовані види робіт**
Впрацьовування	8:00–9:30	Аналіз поставлених завдань, перегляд коду попереднього дня, планування етапу розробки
Стійка висока працездатність	9:30–12:00	Проєктування архітектури, реалізація RAG-пайплайну та LangGraph-агентів, налагодження модуля ChordMini
Неповна компенсація втоми	12:00–13:00	Код-рев'ю, рефакторинг, написання модульних тестів
Обідня перерва	13:00–14:00	Відпочинок, регламентована перерва
Вторинне впрацьовування	14:00–15:30	Розробка інтерфейсу користувача (Jetpack Compose), інтеграційне тестування
Прогресивне зниження працездатності	15:30–17:00	Документування, підготовка звітів, задачі з низьким рівнем відповідальності

На динаміку працездатності оператора, який розробляє програмне забезпечення з використанням технологій штучного інтелекту, негативно впливають такі фактори: тривала статична поза сидячи (понад 6 годин на день), що спричиняє порушення кровообігу та застійні явища в м'язах спини й шиї; підвищене навантаження на зоровий аналізатор унаслідок тривалої роботи з ВДТ, особливо під час перегляду великих обсягів коду та журналів роботи моделей; нервово-емоційна напруга, пов'язана з пошуком і виправленням помилок у розподіленій системі (взаємодія мобільного клієнта, сервера на FastAPI, бази даних Qdrant та зовнішніх AI-сервісів); акустичний дискомфорт від шуму системи охолодження обчислювальної техніки під час локального запуску моделі розпізнавання акордів; недостатня рухова активність протягом робочого дня.

Для збереження високого рівня працездатності та запобігання розвитку перевтоми оператора передбачено: дотримання регламентованих перерв тривалістю 10–15 хв через кожні 45–60 хв безперервної роботи з ВДТ відповідно до вимог ДСанПін 3.3.2.007-98 [29]; виконання комплексів вправ для очей та виробничої гімнастики під час перерв; чергування інтелектуально напружених завдань з розробки AI-компонентів із менш напруженими видами робіт відповідно до фаз динаміки працездатності, наведених у таблиці 4.1; раціональну організацію

робочого місця та оптимальні параметри мікроклімату й освітлення, що розглянуто в підрозділі 4.2.

4.2 Вимоги охорони праці до організації робочого місця користувача персонального комп'ютера

Організація робочого місця оператора, що використовує персональний комп'ютер для розробки програмного забезпечення, регламентується Державними санітарними правилами і нормами роботи з візуальними дисплейними терміналами електронно-обчислювальних машин (ДСанПін 3.3.2.007-98) [28], НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [30], а також стандартами серії ДСТУ з дизайну та ергономіки робочих місць операторів [31–33].

Робоче місце розробника застосунку «Guitarly» розміщене в окремому приміщенні площею 20 м² (5,0 м × 4,0 м), що відповідає нормі не менше 6,0 м² площі та 20,0 м³ об'єму повітря на одне робоче місце з ВДТ [29]. Робочий стіл розташовано перпендикулярно до вікна, що забезпечує природне освітлення без прямого попадання сонячних променів на екран монітора та запобігає виникненню відблисків. Монітор встановлено на відстані від 600 мм до 700 мм від очей оператора, верхній край екрана – на рівні очей або трохи нижче, що відповідає вимогам ДСТУ ISO 9241-5 [33]. Схему організації робочого місця наведено на рисунку 4.2.

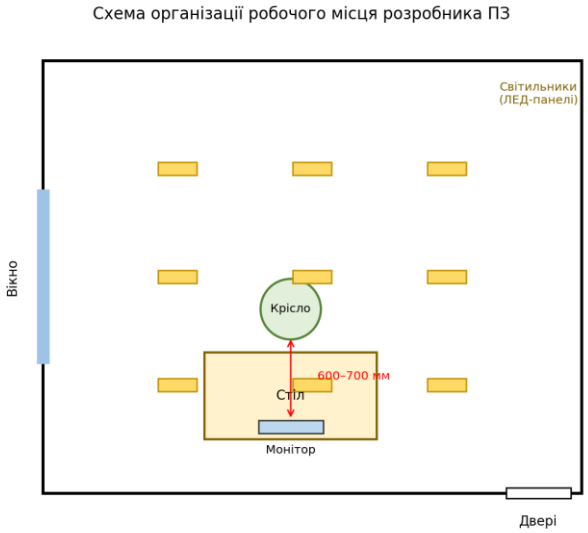


Рисунок 4.2 – Схема організації робочого місця розробника програмного забезпечення

Основні ергономічні параметри робочого місця, прийняті відповідно до нормативних документів, наведено в таблиці 4.2.

Таблиця 4.2 – Ергономічні параметри робочого місця користувача ПК

Параметр	**Нормативне значення**	**Нормативний документ**
Площа на одне робоче місце з ВДТ	не менше 6,0 м²	ДСанПін 3.3.2.007-98
Об'єм приміщення на одне робоче місце	не менше 20,0 м³	ДСанПін 3.3.2.007-98
Висота робочої поверхні стола	680–800 мм (регульована)	ДСТУ 8604:2015
Висота сидіння крісла	400–500 мм (регульована)	ДСТУ 7951:2015
Кут нахилу спинки крісла	90–110°	ДСТУ 7951:2015
Відстань від очей до екрана монітора	600–700 мм	ДСТУ ISO 9241-5:2004
Кут нахилу екрана монітора від вертикалі	0–30°	НПАОП 0.00-7.15-18
Освітленість робочої поверхні (комбіноване)	300–500 лк	ДБН В.2.5-28:2018

Робота з розробки програмного забезпечення за енерговитратами організму належить до категорії Іа (легка фізична робота, енерговитрати до 120 ккал/год) [33]. Оптимальні параметри мікроклімату приміщення для цієї категорії робіт наведено в таблиці 4.3.

Таблиця 4.3 – Оптимальні параметри мікроклімату робочого приміщення (категорія робіт Іа)

Період року	**Температура повітря, °С**	**Відносна вологість, %**	**Швидкість руху повітря, м/с**
Холодний	22–24	40–60	≤ 0,1
Теплий	23–25	40–60	≤ 0,1–0,2

Підтримання зазначених параметрів мікроклімату забезпечується системою кондиціонування повітря, що особливо важливо з огляду на додаткове тепловиділення від високопродуктивної обчислювальної техніки, яка використовується для локального тестування моделі розпізнавання акордів ChordMini.

Розрахунок штучного освітлення приміщення виконано методом коефіцієнта використання світлового потоку [35]. Вихідні дані: довжина приміщення $A = 5{,}0$ м, ширина $B = 4{,}0$ м, площа $S = 20{,}0$ м^2, нормована освітленість для роботи з ВДТ Ен $= 400$ лк (табл. 4.2), розрахункова висота підвісу світильників над робочою поверхнею $H\text{р} = 2{,}2$ м, коефіцієнт запасу $k\text{з} = 1{,}5$, коефіцієнт нерівномірності освітлення $Z = 1{,}1$.

Індекс приміщення визначається за формулою:

$$i = \frac{(A \cdot B)}{H\text{р} \cdot (A + B)} = (5{,}0 \cdot 4{,}0) \,/\, (2{,}2 \cdot (5{,}0 + 4{,}0) = 20{,}0/19{,}8 \approx 1{,}0, \qquad (4.1)$$

де A – довжина приміщення, м;

B – ширина приміщення, м;

$H_\text{р}$ – розрахункова висота підвісу світильників над робочою поверхнею, м.

За значенням індексу приміщення $i \approx 1{,}0$ та орієнтовних коефіцієнтів відбиття стелі, стін і підлоги (70 %, 50 % та 30 % відповідно) коефіцієнт використання світлового потоку приймається $\eta = 0{,}5$ [35].

Необхідну кількість світильників визначено за формулою:

$$N = \frac{(E_н \cdot S \cdot k_з \cdot Z)}{(F_л \cdot n \cdot \eta)} = \frac{(400 \cdot 20{,}0 \cdot 1{,}5 \cdot 1{,}1)}{(4000 \cdot 1 \cdot 0{,}5)} \cdot 13200/2000 \approx 6{,}6, \quad (4.2)$$

де $E_н$ – нормована освітленість для роботи з ВДТ, лк;

S – площа приміщення, м²;

$k_з$ – коефіцієнт запасу;

Z – коефіцієнт нерівномірності освітлення;

$F_л$ – світловий потік одного світильника, лм;

n – кількість ламп у світильнику;

η – коефіцієнт використання світлового потоку.

Отже, для забезпечення нормованої освітленості робочого місця приймаємо до встановлення 7 світлодіодних світильників типу LED-панель 600 мм × 600 мм, рівномірно розміщених по стелі приміщення (рисунок 4.2).

Персональний комп'ютер, монітор, мережеве обладнання та додаткова обчислювальна техніка для запуску AI-моделей живляться від мережі змінного струму напругою 220 В, частотою 50 Гц і за способом захисту людини від ураження електричним струмом належать до класу I електроустановок. Приміщення, в якому організовано робоче місце, за класифікацією ПУЕ [36] належить до приміщень без підвищеної небезпеки ураження електричним струмом (сухе, непилове, з нетоковідними підлогами, нормальною температурою повітря). Для забезпечення електробезпеки передбачено: занулення (заземлення) корпусів ПК та периферійного обладнання; використання розеткових груп із пристроями захисного відключення (ПЗВ) з номінальним струмом спрацювання не більше 30 мА; застосування мережевих фільтрів та джерел безперебійного живлення для захисту чутливого обладнання, яке використовується для тренування та запуску моделей машинного навчання, від перепадів напруги.

За вибухопожежною та пожежною небезпекою приміщення з персональними комп'ютерами відповідно до ДСТУ Б В.1.1-36:2016 [37] належить до категорії В (пожежонебезпечна) через наявність горючих матеріалів – ізоляції кабелів, корпусів пристроїв, меблів, паперових носіїв. Для гасіння можливих осередків

пожежі електрообладнання передбачено вуглекислотні вогнегасники типу ВВК-2 з розрахунку не менше одного вогнегасника на кожні $20\,м^2$ площі приміщення. Шляхи евакуації та двері приміщення відповідають вимогам щодо вільного та безперешкодного виходу персоналу.

Відповідно до ДСанПін 3.3.2.007-98 [29], робота з розробки програмного забезпечення з інтенсивним використанням ВДТ (зчитування та аналіз великих обсягів коду, документації та результатів роботи мовних моделей) відповідає категорії напруженості ІІІ. Для працівників цієї категорії передбачено регламентовані перерви загальною тривалістю 70 хв на робочу зміну: тривалістю 15 хв через 1–2 год після початку зміни та через 1,5–2 год після обідньої перерви, а також додаткові мікропаузи тривалістю 1–2 хв через кожні 10–15 хв безперервної роботи.

4.3 Висновки до розділу 4

У розділі проаналізовано умови праці оператора під час розробки мобільного застосунку «Guitarly» для навчання гри на гітарі з використанням технологій штучного інтелекту.

Розглянуто фазову динаміку працездатності оператора протягом робочого дня (підрозділ 4.1) та запропоновано розподіл видів робіт з розробки застосунку (проєктування архітектури, реалізація RAG-пайплайну, тестування модуля ChordMini, оформлення документації) відповідно до фаз стійкої високої та зниженої працездатності, що сприяє зменшенню кількості помилок і підвищенню якості програмного коду.

Обґрунтовано вимоги до організації робочого місця користувача ПК відповідно до ДСанПін 3.3.2.007-98, НПАОП 0.00-7.15-18 та стандартів ДСТУ з ергономіки (підрозділ 4.2): визначено площу та об'єм приміщення, параметри робочого стола та крісла, відстань до монітора.

Виконано розрахунок штучного освітлення робочого приміщення методом коефіцієнта використання світлового потоку, за результатами якого визначено

необхідну кількість світильників – 7 LED-панелей 600 мм × 600 мм потужністю 36 Вт, що забезпечує нормовану освітленість 400 лк.

Визначено параметри мікроклімату приміщення для категорії робіт Ia, заходи з електробезпеки (занулення обладнання, застосування ПЗВ зі струмом спрацювання до 30 мА) та пожежної безпеки (категорія В, вуглекислотні вогнегасники ВВК-2).

Встановлено регламент праці та відпочинку для категорії напруженості III – регламентовані перерви загальною тривалістю 70 хв на зміну.

Виконання наведених рекомендацій забезпечує збереження здоров'я та працездатності розробника, знижує ризик професійних захворювань опорно-рухового апарату та зорового аналізатора, а також сприяє підвищенню ефективності розробки програмного забезпечення.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено мобільний застосунок "Guitarly" для самостійного навчання гри на гітарі, що поєднує бібліотеку акордів із синхронізованим відображенням під час відтворення відео, інтерактивний AI-тьютор на основі архітектури RAG та автоматичне розпізнавання акордів із YouTube-відео за допомогою моделі ChordMini. Мету та завдання роботи виконано в повному обсязі.

У першому розділі проаналізовано предметну область самостійного навчання гри на музичних інструментах за допомогою мобільних застосунків, оглянуто наявні рішення (Yousician, Ultimate Guitar, Fender Play, Chordify, GuitarTuna) та виявлено їхні обмеження – відсутність інтерактивного AI-супроводу та автоматичного розпізнавання акордів із відео. Обґрунтовано вибір технологічного стеку: Kotlin та Jetpack Compose для Android-клієнта, FastAPI для серверної частини, LangChain та LangGraph для побудови AI-тьютора, Qdrant як векторної бази даних та хмарного провайдера Groq для високошвидкісного інференсу LLM у RAG-конвеєрі.

У другому розділі спроектовано та реалізовано архітектуру системи "Guitarly" за принципами Clean Architecture з розділенням на шари domain, data та presentation. Розроблено серверну частину на FastAPI з підсистемами автентифікації, аналізу YouTube-відео, бібліотеки акордів та AI-тьютора, що використовує LangGraph-граф зі стратегією Self-RAG для генерації контекстно релевантних відповідей у реальному часі через протокол Server-Sent Events. Реалізовано Android-застосунок з модулями тюнера, метронома, синхронізованого відтворення акордів, транспонування та спільноти (challenges).

У третьому розділі здійснено комплексну верифікацію розробленої системи: складено та виконано 18 тестових сценаріїв, усі з яких отримали статус "Пройдено". Проведено навантажувальне тестування серверних API засобами Locust, яке підтвердило стабільну роботу основних ендпоінтів та коректне спрацювання механізмів обмеження частоти запитів (rate limiting) для

ресурсомістких AI-операцій аналізу відео та діалогу з тьютором. Описано процедуру розгортання серверної частини за допомогою Docker Compose та підготовку Android-застосунку до публікації в Google Play Store.

Практичне значення роботи полягає у створенні готового до використання інструменту, що поєднує функції тюнера, бібліотеки акордів і персоналізованого AI-репетитора в єдиному застосунку, що знижує поріг входу для початківців у самостійному навчанні гри на гітарі. Перспективами подальшого розвитку є розширення бібліотеки підтримуваних інструментів, додавання розпізнавання гри користувача через мікрофон у реальному часі для оцінки точності виконання, а також масштабування серверної інфраструктури для збільшення кількості одночасних користувачів AI-тьютора.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Михалик Д. М., Цуприк Г. Б., Бревус В. М. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти за освітньо-професійною програмою «Інженерія програмного забезпечення» спеціальності 121 – «Інженерія програмного забезпечення» всіх форм навчання. Тернопіль : ТНТУ ім. І. Пулюя, 2024. 45 с.
2. Mobile Learning Market Size, Share | CAGR of 24%. Market.us. [Електронний ресурс] URL: https://market.us/report/mobile-learning-market/ (дата звернення: 10.06.2026).
3. How Many People in the World Play Guitar? Breakthrough Guitar. [Електронний ресурс] URL: https://breakthroughguitar.com/how-many-people-in-the-world-play-guitar/ (дата звернення: 10.06.2026).
4. Lewis P., Perez E., Piktus A. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401. [Електронний ресурс] URL: https://arxiv.org/abs/2005.11401 (дата звернення: 10.06.2026).
5. Asai A., Wu Z., Wang Y. et al. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. arXiv:2310.11511. [Електронний ресурс] URL: https://arxiv.org/abs/2310.11511 (дата звернення: 10.06.2026).
6. ChordMiniApp: Music Analysis, Chord Recognition, Beat Tracking. [Електронний ресурс] URL: https://github.com/ptnghia-j/ChordMiniApp (дата звернення: 10.06.2026).
7. Material Design 3 – Google’s open source design system. [Електронний ресурс] URL: https://m3.material.io/ (дата звернення: 10.06.2026).
8. Yousician: Learn Guitar, Piano, Ukulele With The Songs You Love. [Електронний ресурс] URL: https://yousician.com/ (дата звернення: 10.06.2026).
9. Ultimate Guitar – Tabs, Chords, Guitar Pro. [Електронний ресурс] URL: https://www.ultimate-guitar.com/ (дата звернення: 10.06.2026).

10. Fender Play – Online Guitar Lessons. [Електронний ресурс] URL: https://www.fender.com/play (дата звернення: 10.06.2026).

11. Chordify – Learn and Play All Your Favorite Songs. [Електронний ресурс] URL: https://chordify.net/ (дата звернення: 10.06.2026).

12. GuitarTuna – Online Guitar Tuner, Chords & Songs Library. [Електронний ресурс] URL: https://guitartuna.com/ (дата звернення: 10.06.2026).

13. From Superficial Outputs to Superficial Learning: Risks of Large Language Models in Education. arXiv:2509.21972. [Електронний ресурс] URL: https://arxiv.org/abs/2509.21972 (дата звернення: 10.06.2026).

14. Songsterr vs Ultimate Guitar: An In-Depth Comparison. Singular Sound. [Електронний ресурс] URL: https://www.singularsound.com/blogs/news/songsterr-vs-ultimate-guitar-an-in-depth-comparison (дата звернення: 10.06.2026).

15. Examining the effectiveness of gamification as a tool promoting teaching and learning in educational settings: a meta-analysis. Frontiers in Psychology. [Електронний ресурс] URL: https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2023.1253549/full (дата звернення: 10.06.2026).

16. API Levels – Android versions, SDK/API levels, version codes, codenames, and cumulative usage. [Електронний ресурс] URL: https://apilevels.com/ (дата звернення: 10.06.2026).

17. Kotlin and Android. Android Developers. [Електронний ресурс] URL: https://developer.android.com/kotlin (дата звернення: 10.06.2026).

18. Jetpack Compose UI App Development Toolkit. Android Developers. [Електронний ресурс] URL: https://developer.android.com/compose (дата звернення: 10.06.2026).

19. Benchmarks. FastAPI. [Електронний ресурс] URL: https://fastapi.tiangolo.com/benchmarks/ (дата звернення: 10.06.2026).

20. Introduction. LangChain Docs. [Електронний ресурс] URL: https://python.langchain.com/docs/introduction/ (дата звернення: 10.06.2026).

21. Self-Reflective RAG with LangGraph. LangChain Blog. [Електронний ресурс] URL: https://www.langchain.com/blog/agentic-rag-with-langgraph (дата звернення: 10.06.2026).

22. Groq – Fast AI Inference (LPU). [Електронний ресурс] URL: https://groq.com/ (дата звернення: 10.06.2026).

23. Martin R. C. The Clean Architecture. The Clean Code Blog. [Електронний ресурс] URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (дата звернення: 10.06.2026).

24. Server-sent events. MDN Web Docs. [Електронний ресурс] URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events (дата звернення: 10.06.2026).

25. bcrypt. Wikipedia. [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Bcrypt (дата звернення: 10.06.2026).

26. Locust – A Modern Load Testing Framework. [Електронний ресурс] URL: https://locust.io/ (дата звернення: 10.06.2026).

27. Жидецький В.Ц. Охорона праці користувачів комп’ютерів : підручник. Львів : Афіша, 2020. 176 с.

28. Безпека життєдіяльності та охорона праці : підруч. / В.В. Сокуренко, О.М. Бандурка та ін. – Харків : ХНУВС, 2021. 308 с.

29. ДСанПін 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин.

30. НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями : затв. наказом Мінсоцполітики України від 14.02.2018 № 207.

31. ДСТУ 7299:2013. Дизайн і ергономіка. Робоче місце оператора. Взаємне розташування елементів робочого місця. Загальні вимоги ергономіки.

32. ДСТУ 8604:2015. Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги.

33. ДСТУ ISO 9241-5:2004. Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 5. Вимоги до компонування робочого місця та до робочої пози (ISO 9241-5:1998, IDT).

34. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень.

35. ДБН В.2.5-28:2018. Природне і штучне освітлення. Київ : Мінрегіон України, 2018. 133 с.

36. Правила улаштування електроустановок : ПУЕ. Харків : Форт, 2017. 760 с.

37. ДСТУ Б В.1.1-36:2016. Визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою.

ДОДАТКИ

ДОДАТОК А

Діаграми варіантів використання

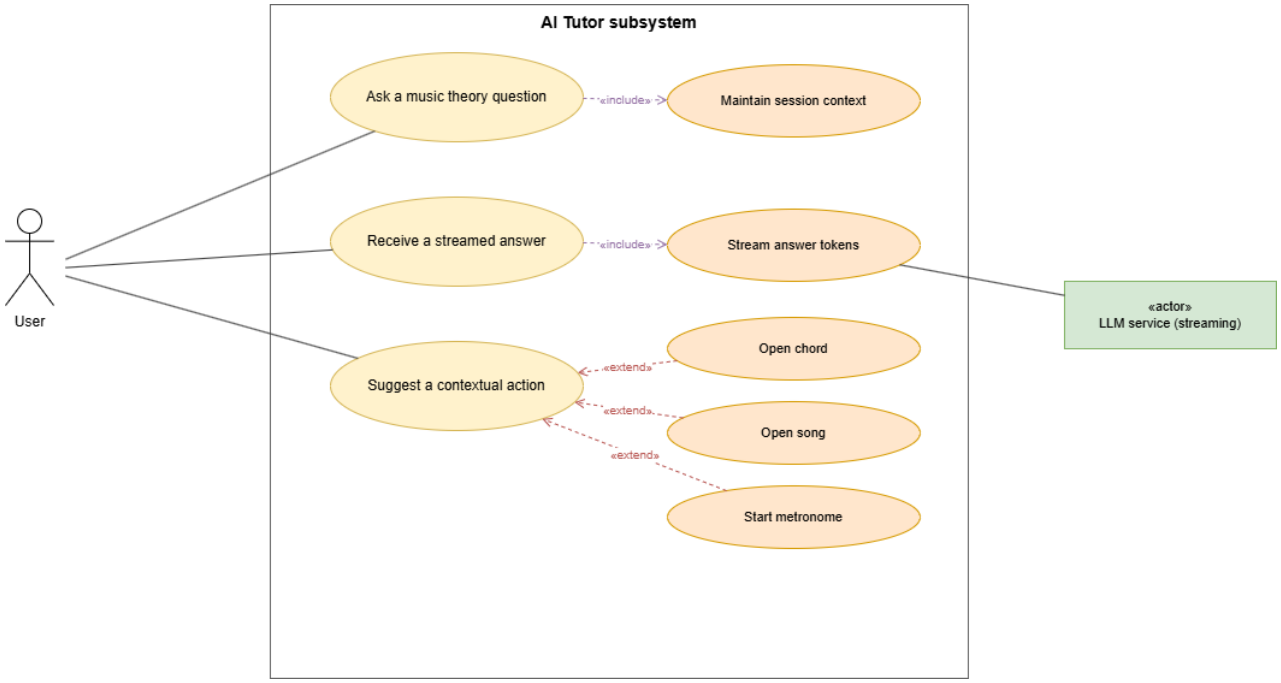


Рисунок 1 – Діаграма варіантів використання AI-тьютора з підтримкою стрімінгу (ФВ-1)

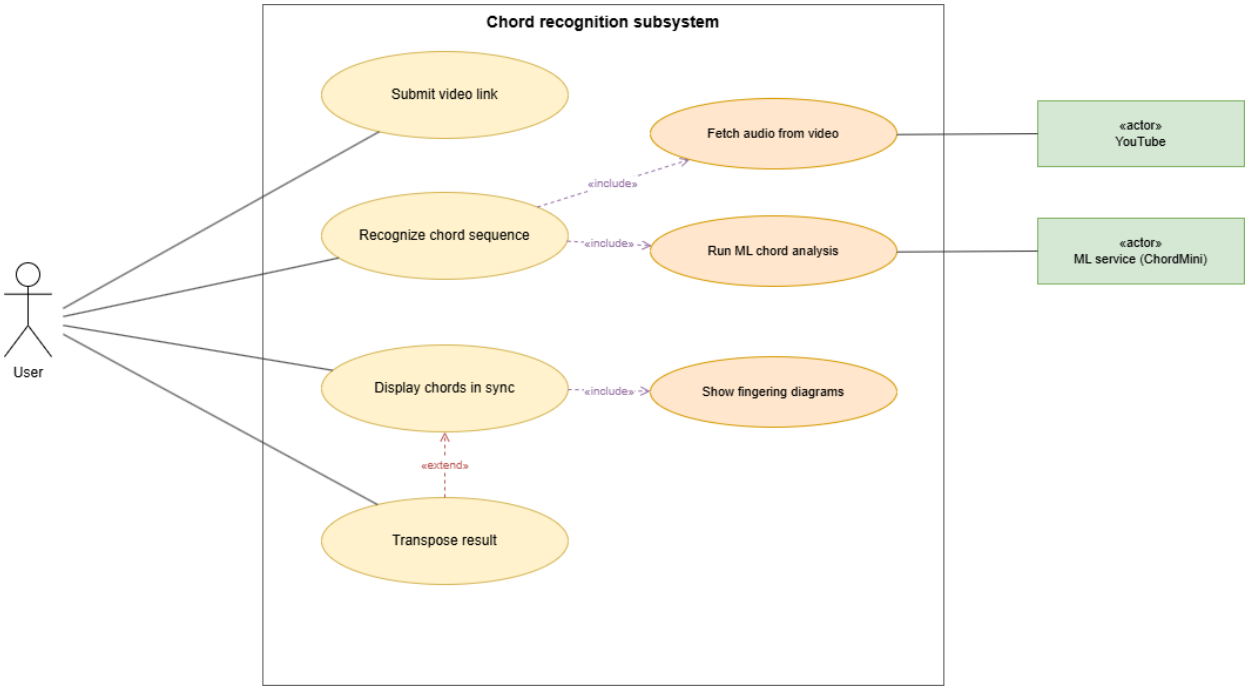


Рисунок 2 – Діаграма варіантів використання розпізнавання акордів у YouTube-відео (ФВ-2)

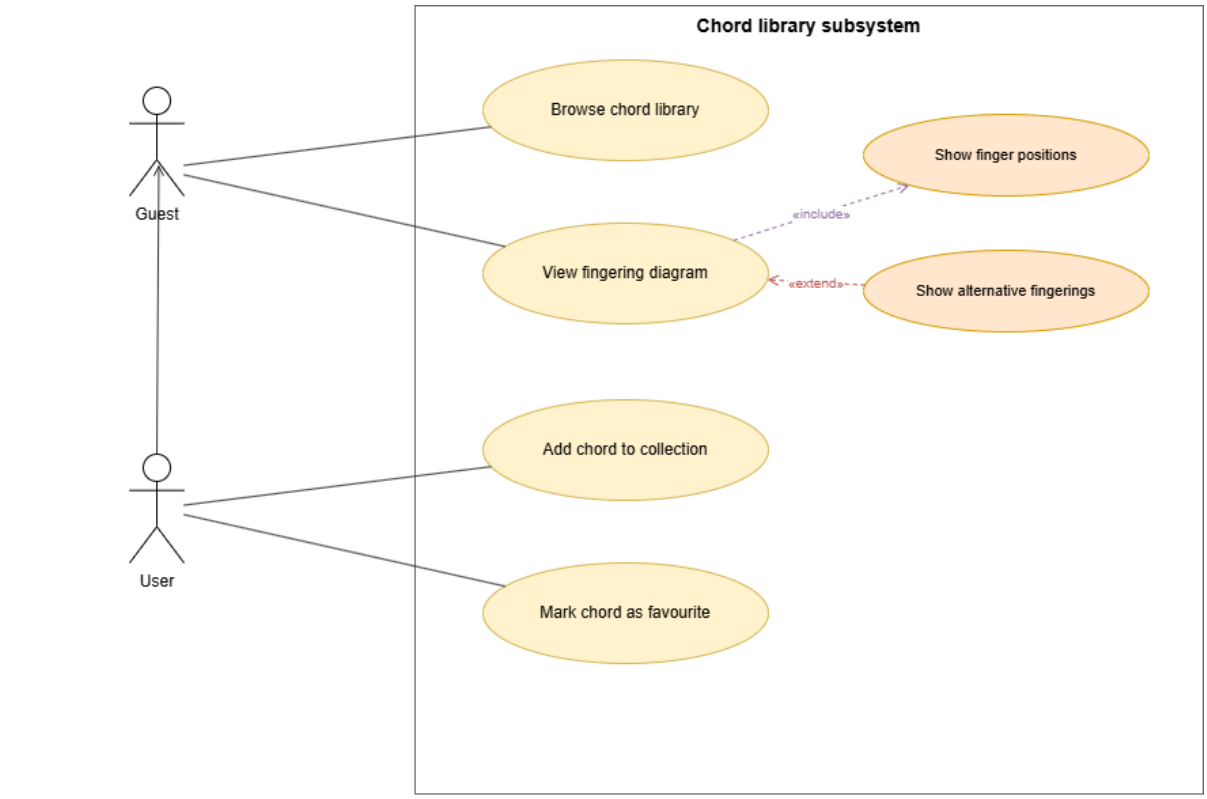


Рисунок 3 – Діаграма варіантів використання бібліотеки акордів із графічними діаграмами (ФВ-3)

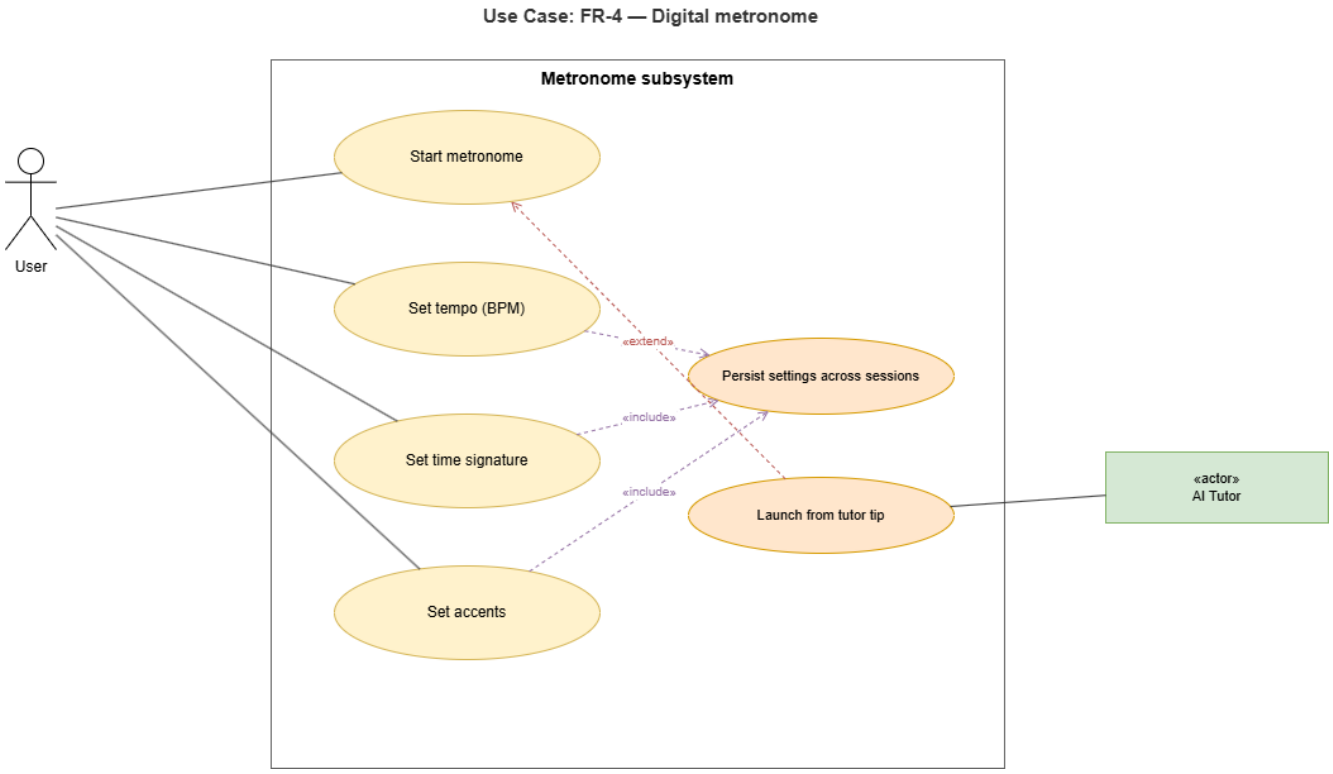


Рисунок 4 – Діаграма варіантів використання цифрового метронома (ФВ-4)

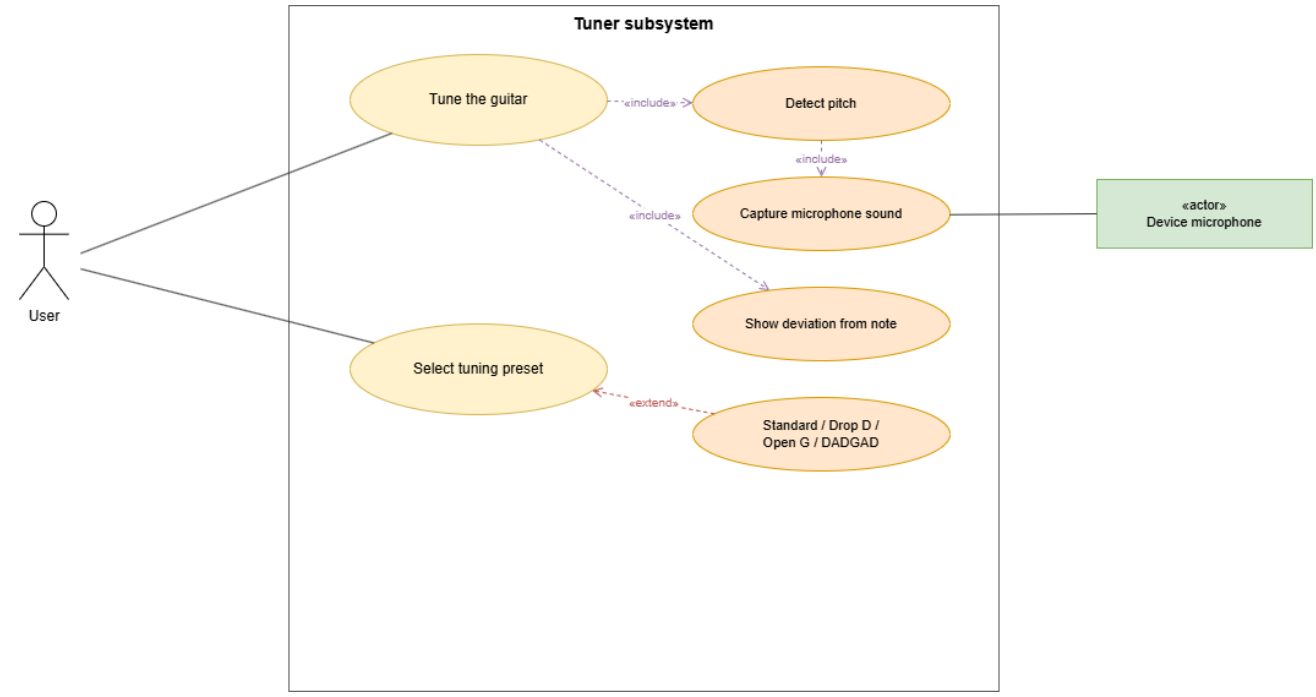


Рисунок 5 – Діаграма варіантів використання хроматичного тюнера (ФВ-5)

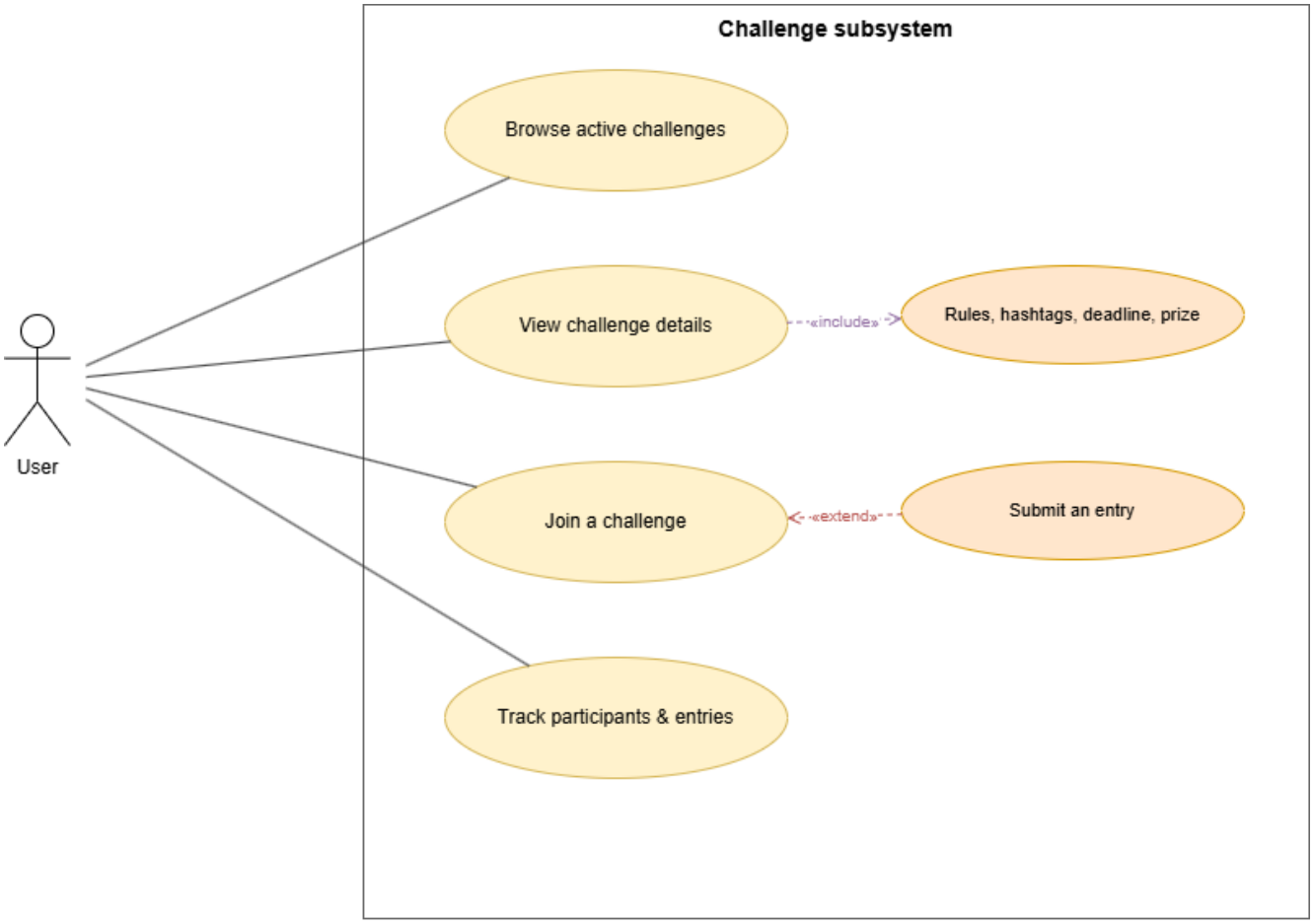


Рисунок 6 – Діаграма варіантів використання системи челленджів (ФВ-6)

ДОДАТОК Б

Тези конференції

IX Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 621.326
Чайківський С. – гр. СП-42
Тернопільський національний технічний університет імені Івана Пулюя

МЕТОДИ РОЗПІЗНАВАННЯ МУЗИЧНИХ АКОРДІВ ЗАСОБАМИ МАШИННОГО НАВЧАННЯ

Науковий керівник: к.ф.-м. н., доцент Цебрій О.Р.

Chaikivskyi S.
Ternopil Ivan Puluj National Technical University

METHODS OF MUSICAL CHORD RECOGNITION USING MACHINE LEARNING

Supervisor: Candidate of Physical and Mathematical Sciences, Associate Professor Tsebrii O.R.

Ключові слова: штучний інтелект, розпізнавання акордів, машинне навчання
Keywords: artificial intelligence, chord recognition, machine learning

Автоматичне розпізнавання музичних акордів є важливою задачею в галузі Music Information Retrieval (MIR), яка знаходить практичне застосування у навчальних застосунках та інструментах для музикантів. Офіційна транскрипція акордів важкодоступна у вільному доступі, оскільки вона не завжди публікується, а якщо й публікується, то нерідко складається вручну непрофесійними музикантами. Тому автоматизація цього процесу засобами машинного навчання є актуальною як для науки, так і для практики [2].

Традиційні підходи до розпізнавання акордів базуються на обчисленні хромаграми, тобто частотно-часового представлення аудіосигналу, де кожен вектор відображає активність 12 класів висоти тону. Для її отримання застосовується перетворення Фур'є (STFT) або константне Q-перетворення (CQT), причому останнє є популярнішим завдяки кращій частотній роздільній здатності [2]. Паралельно широко використовуються мел-частотні кепстральні коефіцієнти (MFCC), які ефективно моделюють сприйняття звуку людиною і слугують компактним описом спектральних характеристик аудіосигналу [4].

Сучасні підходи спираються на глибоке навчання. Ранні моделі використовували рекурентні (RNN) та згорткові нейронні мережі (CNN), а пізніше з'явилися гібридні CRNN-архітектури, що поєднують переваги обох підходів [2]. Серед різних архітектур таких як CNN, AlexNet, VGG-19 та ResNet-50 – модель на основі спектрограми демонструє найкращі результати в задачі класифікації гітарних акордів, ефективно вловлюючи висотні та гармонічні залежності [4]. Важливим інструментом підвищення стійкості моделей є аудіо-аугментація: додавання шуму, зміна швидкості та реверберація, що суттєво покращують точність класифікації на реальних даних [1]. Одним із практично застосовних рішень є модель Basic Pitch, розроблена Spotify Audio Intelligence Lab і представлена на конференції ICASSP 2022. Вона призначена для конвертації аудіозаписів у формат MIDI за допомогою легковагової нейронної мережі та здатна обробляти поліфонічний звук, виявляючи одночасно виконувані ноти, pitch bend, точні тайминги атак і тривалості нот [3]. Basic Pitch є ресурсоефективною і здатна працювати швидше, ніж у реальному часі на більшості сучасних комп'ютерів [5].

Завдяки підтримці формату TensorFlow Lite модель може бути розгорнута безпосередньо на мобільних пристроях без потреби у серверній обробці [5]. На датасеті GuitarSet модель досягає F-міри на рівні 79% для гітарних партій, що є конкурентоспроможним результатом серед легковагових систем транскрипції [3].

Таким чином, поєднання класичних методів виділення ознак (хромаграма, CQT, MFCC) із сучасними архітектурами глибокого навчання та їх адаптація під мобільні платформи відкриває широкі можливості для створення точних і практично застосовних систем автоматичного розпізнавання акордів у реальному часі.

Література:

1. Пірко А., Борецька І. Методи аудіо-аугментації у моделях машинного навчання // Вісник ХНУ. Технічні науки. — 2024. — Т. 341, № 5.
2. Boulanger-Lewandowski N., Bengio Y., Vincent P. Audio Chord Recognition with Recurrent Neural Networks // ISMIR. — 2013. — P. 335–340.
3. Bittner R. M., Bosch J. J., Rubinstein D., Meseguer-Brocal G., Ewert S. A Lightweight Instrument-Agnostic Model for Polyphonic Note Transcription and Multipitch Estimation // ICASSP. — 2022.
4. Korade N. B., Salunke M. B., Bhosle A. A. та ін. Intelligent Guitar Chord Recognition Using Spectrogram-Based Feature Extraction and AlexNet Architecture for Categorization // IJACSA. — 2025. — Vol. 16, No. 4.
5. Bittner R. M. Meet Basic Pitch: Spotify's Open Source Audio-to-MIDI Converter // Spotify Engineering Blog. — 2022. — URL: engineering.atspotify.com/2022/06/meet-basic-pitch

ДОДАТОК В

Clean Architecture шари

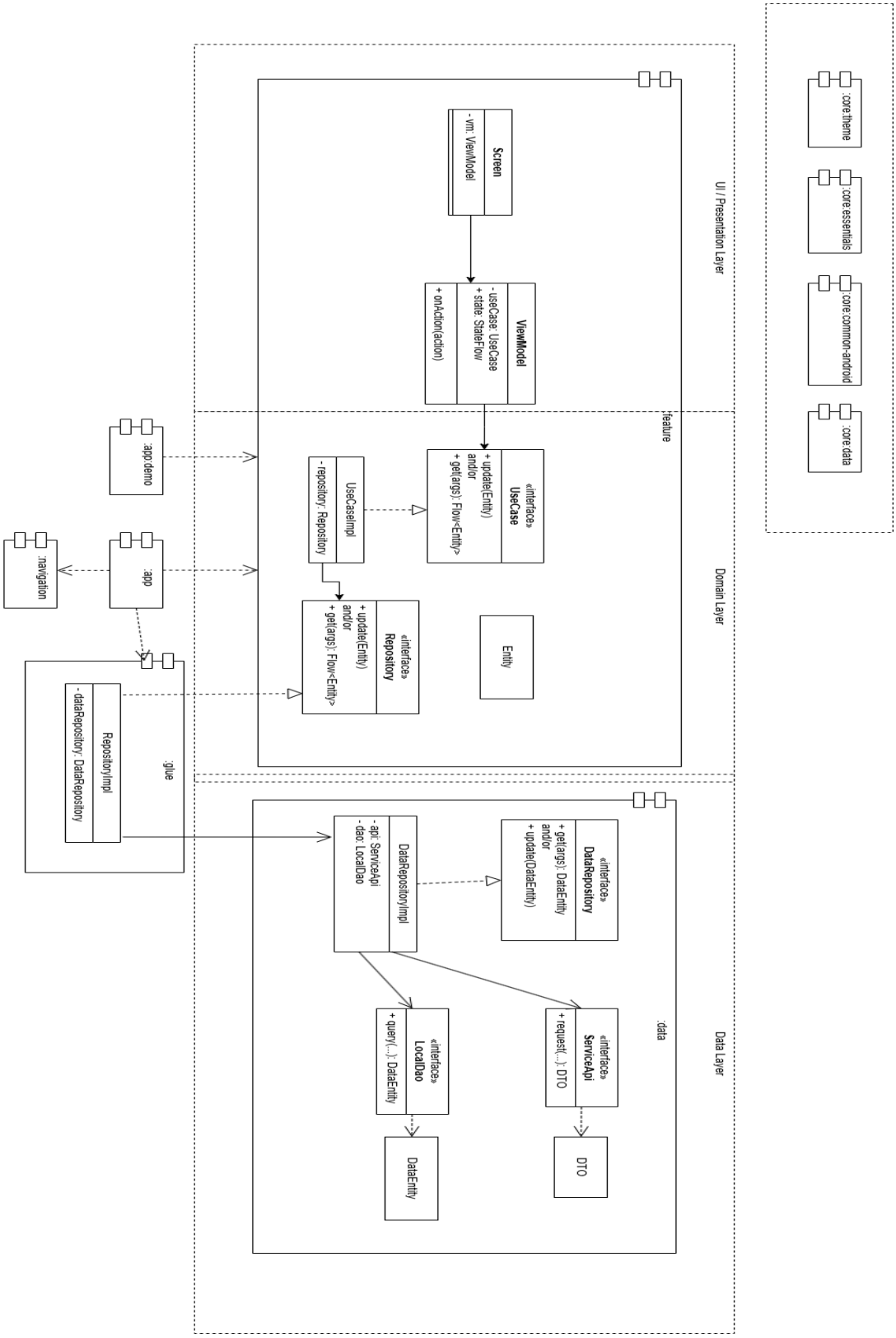


Рисунок 1 – Clean Architecture шари застосунку "Guitarly"

ДОДАТОК Г

Тестові сценарії системи "Guitarly"

Таблиця 1 – Тестові сценарії системи "Guitarly"

№	Сценарій тестування	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
1	2	3	4	5	6
1	Реєстрація нового користувача з валідними даними	Email: test@example.com, пароль: Test1234!	Обліковий запис створено, перехід на головний екран	Обліковий запис успішно створено, виконано перехід на головний екран	**Пройдено**
2	Вхід у систему з некоректним паролем	Email: test@example.com, пароль: wrongpass	Повідомлення про помилку авторизації	Виведено повідомлення "Невірний email або пароль"	**Пройдено**
3	Введення валідного YouTube URL та запуск аналізу акордів	https://www.youtube.com/watch?v=dQw4w9WgXcQ	Задача аналізу поставлена у чергу, відображається індикатор прогресу	Задача аналізу зареєстрована, відображається статус "Обробляється"	**Пройдено**
5	Синхронізація акордів із відтворенням YouTube-плеєра	Пісня з 10 акордами, поточний час: 2.5 с	Виділяється акорд, що відповідає часовій мітці 2.5 с	Акорд на позиції 2.0 с підсвічено; при переході до 4.0 с підсвічується наступний	**Пройдено**
6	Транспонування акордів на +2 півтони	Акорд C – транспозиція +2	Відображається акорд D	Відображається акорд D	**Пройдено**
7	Транспонування акорду Am на -1 півтон	Акорд Am – транспозиція -1	Відображається акорд G#m або Abm	Відображається акорд G#m	**Пройдено**
8	Надсилання запиту до AI-тьютора	Повідомлення: "Як навчитися грати акорд G?"	AI-тьютор повертає відповідь з поясненням та цитатами із бази знань	Отримано текстову відповідь у режимі стрімінгу з посиланнями на джерела	**Пройдено**

Продовження таблиці 1

1	2	3	4	5	6
9	Запит до AI-тьютора без мережевого з'єднання	Повідомлення: "Як взяти акорд Bm?" при відсутності інтернету	Повідомлення про відсутність з'єднання	Відображається повідомлення про помилку мережі	**Пройдено**
10	Пошук акорду за назвою у бібліотеці	Запит: "Am"	Відображається картка акорду Am з діаграмою постановки пальців	Знайдено акорд Am, відображено діаграму та аплікатуру	**Пройдено**
11	Фільтрація акордів за складністю	Фільтр: "Початківець"	Відображаються лише акорди рівня Beginner	Список відфільтровано: відображено Am, C, D, E, Em, G	**Пройдено**
12	Відображення альтернативних аплікатур акорду	Акорд D, запит альтернатив	Відображається 2–3 альтернативні варіанти постановки	Відображено 3 варіанти аплікатури акорду D	**Пройдено**
13	Запуск тюнера та визначення ноти	Звук ноти A (440 Гц) через мікрофон пристрою	Тюнер відображає ноту A4, відхилення ±0 cents	Відображено ноту A4, відхилення −2 cents (в межах точності)	**Пройдено**
14	Запуск метронома зі значенням 120 BPM	BPM: 120, розмір: 4/4	Метроном відтворює 120 ударів на хвилину з акцентом на першому	Метроном запущено, удари рівномірні, акцент на долі 1 підтверджено	**Пройдено**
15	Зміна темпу метронома повзунком	Переміщення повзунка з 120 до 80 BPM	Темп змінюється без зупинки метронома	Темп плавно знизився до 80 BPM без паузи у відтворенні	**Пройдено**
16	Перегляд списку доступних завдань (challenges)	Користувач авторизований, відкривається екран Challenges	Відображається список завдань із назвою, автором та датою	Список завдань завантажено та відображено коректно	**Пройдено**
17	Додавання коментаря до завдання	Текст коментаря: "Чудовий набір акордів!"	Коментар додано і відображається у списку	Коментар збережено та відображено під завданням	**Пройдено**